Architectures based on Oblivious RAM for Enhancing User Privacy and their Applications to Genome Processing

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation von Nikolaos P. Karvelas aus Athen
Tag der Einreichung: 17. Mai 2018, Tag der Prüfung: 28. Juni 2018
Darmstadt — D 17

1. Gutachten: Prof. Dr. Stefan Katzenbeisser
2. Gutachten: Prof. Dr. Marc Fischlin

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Security Engineering

Architectures based on Oblivious RAM for Enhancing User Privacy and their Applications to Genome Processing

Genehmigte Dissertation von Nikolaos P. Karvelas aus Athen

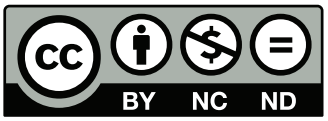
1. Gutachten: Prof. Dr. Stefan Katzenbeisser
2. Gutachten: Prof. Dr. Marc Fischlin

Tag der Einreichung: 17. Mai 2018
Tag der Prüfung: 28. Juni 2018

Darmstadt — D 17

Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-75737
URL: http://tuprints.ulb.tu-darmstadt.de/7573

Dieses Dokument wird bereitgestellt von tuprints,
E-Publishing-Service der TU Darmstadt
http://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de



Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 17. Mai 2018

(Nikolaos P. Karvelas)

Zusammenfassung

Das ständig wachsende Bedürfnis nach Schutz digitaler Daten hat die Entwicklung einer Vielfalt kryprographischer Primitiven als Folge, die die Privatheit der Nutzer garantieren sollen. Nichtsdestotrotz wurde sehr früh deutlich, dass der bloße Schutz des Dateninhalts in vielen Szenarien nicht ausreicht, da die Art und Weise in der auf die Daten zugegriffen wird sehr wichtige Information verrät, die zum Reverse-Engineering von Programmen und zum kompletten Zusammenbruch der Nutzernprivatheit führen kann, wenn Nutzer ihre Daten auf entfernten Server auslagern.

Um dieses Problem zu lösen, hat die kryprographische Gemeinschaft mehrere Primitiven vorgeschlagen, unter welchen Oblivious RAM (ORAM) ein sehr prominenter Ansatz ist, der die Privatheit der Daten auf entfernten Server garantiert. Das dominanteste Forschungsmodel ist das ein Client/ ein Server, welches einerseits für die abstrakte Betrachtung des Problems notwendig ist, andererseits aber wegen der starken Beschränkungen für praktische Probleme schwer anwendbar ist.

In dieser Arbeit entwickeln wir ORAM-Architekturen, welche diese Problematik adressieren und wir verallgemeinern diese Untersuchung. Insbesondere schlagen wir Lösungen vor, die mehreren Klienten erlauben, ihre Daten auf entfernten Server zu speichern und Teile ihrer Daten miteinander oder selektiv mit dritten Parteien zu teilen. Dementsprechend stellen wir die Studie der „Partially Sharing Multi-Client ORAMs“ vor und präsentieren konkrete Instanzen davon. Um die Sicherheit unserer Konstruktionen zu beweisen, war es notwendig, ein neues Framework zu entwickeln, welches uns die Möglichkeit gibt, über die Sicherheitsgarantien solcher komplexen Architekturen zu argumentieren. Darüberhinaus haben wir ein sehr versatiles Framework für ORAM-Sicherheit entwickelt, welches deutlich strenger, als die aktuell benutzten ist, während es gleichzeitig viel einfacher als das von Goldreich und Ostrovsky vorgeschlagene Framework ist. Unter diesem Licht sind wir der Meinung, dass unser Framework hilfreich für die Durchführung von Sicherheitsbeweisen in neuen ORAM-Konstruktionen sein kann.

Um die Anwendbarkeit unserer Architekturen zu zeigen, haben wir sie implementiert und für die Lösung des Problems „Privacy Preserving Genomic Studies“ eingesetzt, welches unserer Meinung nach eine sehr wichtige Rolle in der weiteren Digitalisierung unserer Gesellschaft spielen wird. Mit unseren neuentwickelten Techniken können wir verschlüsselte, sequinzierte Genomdaten mehrerer Klienten speichern, auf sie zugreifen, sie aktualisieren und sie weiter durch Nutzung von „Secure Computation“ Techniken verarbeiten, um eine Vielfalt von Tests durchführen zu können, die sich von einfachen DNA-Fingerprinting zu komplexeren Genomeweit Assoziierungs Studien erstrecken.

Abstract

The all increasing need for data protection has given raise to a plethora of privacy preserving cryptographic techniques that address this issue. Early on however, it was made clear that in many cases, simply protecting the contents of the data is not enough: The way encrypted data is accessed can leak important information that spans from reverse-engineering of programs, to totally breaking the privacy of users, once they store it on remote servers.

To answer this question, the cryptographic community developed various solutions among which one of the most prominent is Oblivious RAM (ORAM), which guarantees data privacy in remotely outsourced private data. Yet, the dominant model under consideration is that of one client and one server, which although necessary as a means of abstractly dealing with the general problem, still has limited applications in real-world scenarios.

In this work, we develop ORAM architectures that address this problem and generalise the above study. In particular, we propose solutions that allow multiple clients to store their private data on remote servers and either share with each other parts of their data, or selectively give access to parts of their data to third parties. This way, we introduce the study of "Partially Sharing Multi-Client ORAMs" and give concrete instantiations of such ORAMs. Proving that our constructions are secure, required the development of a new framework that allows for arguing on the security of such complex architectures. To this extent, we developed a versatile framework for ORAM security that is more rigorous than the frameworks used in practise today and at the same time far easier to use than the original Goldreich-Ostrovsky framework. Under this light, we believe that our new framework will prove itself to be very useful also in showing security for ORAM constructions that will be developed in the future.

In order to show the applicability of the architectures we developed, we implemented and applied them to a problem domain that we believe will be of great significance in the following years. In particular, we address the problem of privacy preserving genomic studies. Using our newly developed techniques, we can store, access and update encrypted sequenced genomes of multiple clients, and by employing secure computation techniques, we can further process the outsourced data for a variety of tests ranging from simple DNA fingerprinting to more complex Genome Wide Association Studies.

Danksagung

Eine der tiefsten mystischen Lehren, die uns das antike Griechenland geliefert hat, ist die in Delphi zu findende Inschrift *Γνῶθι σεαυτόν*, „Erkenne dich selbst“, welche sich in erster Betrachtung als ein Verfahren verstehen lässt, das in kompletter Einsamkeit durchgeführt werden soll. Eine tiefere Beleuchtung offenbart allerdings, dass das nicht der Fall ist. Ganz im Gegenteil spielt die externe Unterstützung in dem Verfahren des eigenen Wiederentdeckens eine fundamentale Rolle.

Die Entwicklung zum Wissenschaftler wird in besonderer Weise durch den Betreuer begleitet. Auf meinem persönlichen wissenschaftlichen Weg bin ich gesegnet gewesen, Stefan Katzenbeisser als meinen Betreuer zu haben. Stefan hat mir bei jedem Schritt dieses marathonischen Verfahrens zur Erlangung des Doktorgrades geholfen, mir immer die Freiheit gegeben, mich zu entwickeln und wie ein liebevoller Vater immer sichergestellt, dass ich meinen Weg nicht verliere. Seine tiefe Intuition ist immer ein Kompass für mich gewesen. Durch unsere langen Diskussionen konnte ich meine Erkenntnisse und meine Intuition für die Ideen, für die Wissenschaft und für das Ingenieurwesen weiterentwickeln. Für all dies bin ich Stefan zutiefst dankbar und dafür, dass er mich als sein Student angenommen hat.

Die Umgebung eines Vorhabens stellt für dessen Erfolg eine katalytische Rolle dar. Die Technische Universität Darmstadt stellte eine fürsorgliche Umgebung dar, die mir im Rahmen von CYSEC und CROSSING Ruhe und Stabilität gegeben hat und mein Unternehmen vereinfachte. Deswegen möchte ich allen Mitarbeitern dieser Einrichtung dafür danken, dass sie sie zu einer gastfreundlichen und offenen Forschungseinrichtung gestalten, die offen für neue Ideen und Innovationen ist.

Während meiner Arbeit an der Technischen Universität Darmstadt hatte ich auch das Privileg die Wandlung einiger Kollaborationen zu tiefen Freundschaften zu erleben. Tommaso Gagliardoni hat mir geholfen, Einsichten in die Thematik „Provable Security“ zu erwerben. Sebastian Biedermann hat mich die Prinzipien der Hacker-Mentalität gelehrt und André Schaller hat mir durch unsere zahlreichen „Brainstorming Sessions“ geholfen, meine wissenschaftliche Sicht zu schärfen. Tommaso, Sebastian, André und meinen lebenslangen Freunden Joseph Papadatos und Kostas Michalopoulos bin ich für ihre Freundschaft und liebevolle Unterstützung herzlich dankbar.

Dennoch hätte mein Unternehmen ohne die großzügige Unterstützung meiner Frau und Freundin Agne, die während jeder meiner Bestrebungen immer dabei gewesen ist, nicht stattfinden können. Agne hat mir die Schönheit des Lebens gezeigt, mein Sohn Perseus hat mich gelehrt meine Zeit am effektivsten zu nutzen und deswegen bin ich beiden zutiefst dankbar. Schließlich möchte ich meinen Eltern Panaiotis und Maria danken. Es gibt keine ausreichende Art, in der ich meinen Vater für seine Lehre des Ethos und meine Mutter für ihre Lehre der Sympathie belohnen kann. Ich kann nur zutiefst dankbar dafür sein, dass sie mich unbegrenzt unterstützt haben und die Grundlagen dafür gelegt haben, was ich heute bin.

Nikolaos P. Karvelas
Darmstadt, Mai 2018

Acknowledgements

One of the greatest, mystical teachings ancient Greece has delivered us, is the famous Delphi inscription *Γνῶθι σεαυτόν*, "know thyself", which at first seems as a procedure to be performed by an individual on its own. A deeper contemplation however, reveals that this is not the case. On the contrary, external assistance is a fundamental and essential part of the procedure of rediscovering oneself.

In the course of becoming a scientist, this difficult role is assigned to the apprentice's advisor. In my personal scientific path, I have been blessed to have Stefan Katzenbeisser as my advisor. Stefan has helped me in every step of this Marathonian procedure towards becoming a doctor of philosophy. Stefan gave me the freedom of evolving, while like a caring father, always made sure that I would not lose my path. His deep intuition has always been a compass for me, and through our lengthy discussions I was too able to develop my insight and intuition to the ideas, science and engineering. For all these, I am deeply grateful to Stefan for having me accepted as his student and being my advisor.

The surrounding environment plays a catalytic role to the final outcome of every endeavour. Technische Universität Darmstadt has been the nurturing environment that through CYSEC and CROSSING offered a tranquil stability that made my enterprise easier. I would thus like to thank all the people involved into making Technische Universität Darmstadt such a hospitable environment for new ideas and research.

In the course of my work in Technische Universität Darmstadt, I had the privilege of witnessing the transformation of scientific collaboration to friendship. I am grateful to Tommaso Gagliardoni for all the insight he helped me attain in the field of provable security, to Sebastian Biedermann for introducing me to the hacker mentality, and to André Schaller for the numerous brainstorming sessions that helped me clear my scientific view. I am grateful to Tommaso, Sebastian and André along with my life long friends Joseph Papadatos and Kostas Michalopoulos for their friendship and dear support.

Yet, none of this would have been possible without the endless support of my wife and friend Agne, who has been there in every quest that I pursued. I am grateful to Agne for showing me generosity, limitless support and teaching me the importance of life, and to my son Perseus for teaching me how to make best use of my time. Finally, I am grateful to my parents, Panaiotis and Maria. There is no way I can requite my father for his teaching of ethos and my mother for her teaching of sympathia. I can only be deeply grateful for their boundless support and for laying the fundamentals of what I am today.

Nikolaos P. Karvelas
Darmstadt, May 2018

ταὶ Πεληάδες γὰρ ἅμιν
ὀρθρίαι φᾶρος φεροίσαις
νύκτα δι' ἀμβροσίαν ἅτε σήριον
ἄστρον ἀνηρομέναι μάχονται

Alkman, Partheneion-Fragment

G. At. und G. Ap. gewidmet

Contents

List of Figures

List of Tables

Acronyms

PSMC-ORAM Partial Sharing Multi-Client ORAM. 4, 65, 67–70, 72, 73, 106

CvHP Chaum van Heisjt Pfitzmann hash function. 20, 59, 60, 95

DDH Decisional Diffie Hellman. 18, 20, 57, 58

DMZ Demilitarised Zone. 4

MPC Secure Multi-Party Computation. 22

ORAM Oblivious Random Access Machine. 3–5, 7–10, 20, 24, 25, 38, 39, 44, 49, 62, 64–66, 68, 87, 90–92, 96, 106

PET Privacy Enhancing Technology. 1–4, 91, 106

PIR Private Information Retrieval. 3

PKI Public Key Infrastructure. 87

PPT Probabilistic Polynomial Time. 15, 17, 25, 28, 35, 51, 57, 66, 70, 82, 84

SNP Single Nucleotide Polymorphism. 2, 11, 93, 94, 99–102

STC Secure Two-Party Computation. 3, 22, 91, 106

STR Short Tandem Repeat. 11, 93, 94

Chapter 1

Introduction

Although privacy has been an integral part of human societies ranging from the classic antiquity [Bur00] to modern times, the notion of privacy as a human right appeared only a little longer than a century ago in Warren and Brandeis' influential article [WB90], at a time that technological advancements seemed to make the invasion to the private sphere of individuals possible. In today's digital society, the protection of each individual's privacy stands again in the centre of our *physical society's* focus, as the society attempts to define anew and to preserve its citizens' freedom.

The digital revolution that has taken place in the last decades has brought with it radical changes in the ways that individuals perceive and handle their privacy. The solutions to every-day problems that have been offered open handedly by computer systems constantly connected with each other, and the all increasing need for more speed, efficiency, and an exciting 'user experience' have brought with them an overwhelming burden on the user's privacy. To mitigate this problem and to answer the sceptics' concerns, the cryptographic community offered solutions mainly in a twofold way: As individual cryptographic tools (such as encryption schemes, proofs of knowledge, etc.) that can be directly applied to existing solutions, and as architectures *designed with security and privacy in mind*, that combine a variety of cryptographic primitives. The latter are usually referred to as Privacy Enhancing Technologies (PETs), and in the past years they have been a new field of intense research which has proposed solutions that achieve anonymous communication, private access and private processing of outsourced data. As such, PETs can play a significant role in the protection of user privacy, and thus the development of such technologies is of great importance for the further steps of both our digital and physical societies.

However, the development of such technologies is not a trivial task. The design of such PETs faces many, synergistically occurring challenges: For one, the data stored in remote servers spans to sizes of several Terabytes which under encryption results in a blowup in the size of the encrypted data. This in turn has as consequence, heavy communication and computation overheads once this data needs to be processed 'under encryption'. At the same time, it is at the moment unclear how these outsourced data will be used in the future and which queries will be performed on it. Take for example the case of privacy preserving processing of outsourced data, a scenario that involves queries to remote databases. It is not improbable that in such a scenario the need will arise for performing complex statistical queries on outsourced data, which in turn can potentially result in information leakage and privacy loss. Thus, a practical PET should be designed in a way that it can support and protect *any* query in order to adapt to future advances of data

processing. Since cryptographically protected data will likely be stored only once, the form of protection should not be tailored to a specific query type but should be as flexible as possible, in order to be able to cope with future queries, while the data owner should be able to update the outsourced data in a way that does not yield privacy compromises. Finally, an ideal PET solution should be able to hide the *type of the query* from the involved parties, which in many cases can be inferred by the mere observation that certain parts of the data have been accessed more frequently than others.

In the past years, many solutions have been proposed that deal with each of the above problems *individually*. The computational cost of encryption schemes has been dramatically shrunk, with solutions such as the AES-NI [INT] set of instructions or the employment of efficient public key encryption schemes such as the ones based on elliptic curves. The flexibility and support of future queries can be achieved by using recently developed and highly efficient protocols for secure multiparty computation [Fur+17], while hiding the type of the query can be achieved by using privacy-preserving building blocks such as searchable encryption [WWC16] or privacy-preserving protocols such as private information retrieval [Cho+] or oblivious RAM [Gol87].

Current cryptographic methods for secure computation however are tailored towards small and mid-size computational problems, while problems such as genomic processing which today is predominantly done through tests looking for Single Nucleotide Polymorphisms (SNPs) (i.e., checking whether a certain symbol occurred in the genome at a given position) require very large inputs. It is foreseeable (and desirable) that more complex statistical tests will enter the arena [PM+99; Lau+13] – especially as for example SNPs do not explain to the full extent heritable diseases like autism [Cur+11; Seb+07]. Furthermore, association studies where genetic and physiological data are combined pose a great challenge in biostatistics and therefore PETs should support these applications as well. Analysis *across* several genomes must be possible to investigate inheritable traits. Again, autism is an example, where genomic heterogeneity and so-called copy-number variation are known to be disease-related [Abr08]. Additionally even twins can be distinguished [JSA11] based on their respective phenotype which reveal complex, systems-biological effects. It is thus to expect that the inputs of such problems will be so large that secure computation techniques alone will not be able to handle them efficiently.

Therefore, it becomes clear that the way these technologies can be moulded in order to be later synthesised in a solution that holistically addresses all the challenges of privacy enhancing technologies has still not been answered. We do this in this work by answering the following question in a positive way:

Can we build a privacy enhancing technology that provides strong security guarantees and can be applied to contemporary data-intensive problems?

In order to address the above question, we first focus on the way that Random Access Machines function, since the way a program accesses specific memory registers depending on its inputs can leak important information not only about the program itself, but also about the inputs. Suppose for example that a client outsources his private digital data in the form of small encrypted *datablocks* to an untrusted server. Without loss of generality, we can assume that some datablocks will be of greater interest than

others for the client. As a result, these datablocks will be accessed more frequently, something that the server can observe independently of the fact that the datablocks are encrypted. From this observation alone, the server can deduce important information not only about the client's query, but also about the data itself. Consider for example the case where a client stores encrypted electronic health records to a remote server: If the server knows that a particular datablock holds data about a specific disease, and observes a client accesses that datablock more frequently than others, then the server can deduce with a certain probability that the client suffers from that particular disease without having to actually know the unencrypted contents of the client's datablocks.

To address the above security problems, Oblivious Random Access Machine (ORAM) is one of the most prominent cryptographic primitives that can be applied in order to mitigate the above information leakage. Goldreich introduced the ORAM primitive in his seminal work [Gol87], motivated by the problem of software protection. There, hiding a program's access patterns should make the program's reverse engineering impossible. Since its introduction, ORAM has received great attention from the cryptographic community, and not undeservedly: Building a computing machine that delivers the correct result of a computation while being oblivious to its every step is a fascinating endeavour that could eventually guarantee data privacy both during storage as well as during processing. As a result, a plethora of constructions have been developed and intensely optimised. ORAM technologies have found their way in a multitude of applications including secure processors [Maa+13], privacy preserving filesystems [WST12], and privacy preserving storage of electronic health records [Maf+15]. Observe here, that since ORAM *is* a Random Access Machine, it offers both read and write capabilities and this way, the flexibility one gains from using this primitive outperforms that of other technologies such as Private Information Retrieval (PIR), which supports only read operations.

Using the ideas presented in state-of-the-art ORAM solutions, we develop in this work new ORAM solutions that enhance user privacy in a multitude of ways. To do this, we first provide a novel framework for proving the security of ORAM constructions. We first demonstrate our new framework's potential by proving the to-date most efficient ORAM construction (namely Path-ORAM [Ste+13a]) secure – something that in Path-ORAM's original proposition was done at a rather high-level. Our new framework allows us then to develop and prove secure two new ORAMs.

In our first construction, we are motivated by the problem of fully outsourcing the retrieval and storage of data to untrusted parties. Thus, our first solution *Proxy-ORAM* operates on encrypted data in *every step of the protocol*, as opposed to classical ORAM solutions, where particular steps (such as the 'reshuffling' of hierarchical constructions, or the 'eviction' of tree-based constructions) have to be performed on plain data. This way, we give the ability to an untrusted third-party to access only parts of the ORAM and retrieve the result in encrypted form, while the owner of the data does not need to be constantly online. We further couple this ORAM construction with Secure Two-Party Computation (STC), and we achieve a PET that guarantees full privacy of data, not only during computation but also during its retrieval.

In our second construction, we deal with a different problem. Most existing ORAM constructions focus on a *one client-one server* setting, which although very interesting and useful for theoretical purposes,

it is rather limiting when it comes to real world applications. The reason for this is that in most interesting scenarios many clients are involved. Furthermore, although research on multi client ORAMs has been already underway for several years, most of the solutions proposed thus far deal primarily with the problem of client anonymity, i.e., concealing from the server who is the client accessing the ORAM. The problem that has been however neglected is that of information leakage due to data accesses between clients in such a multi-client environment.

In this work, we close the above gap by initiating the study of Partial Sharing Multi-Client ORAMs (PSMC-ORAMs). We extend our new ORAM framework in a way that it can describe and rigorously prove secure an ORAM architecture that supports multiple clients storing their private data on the same server and partially share data with each other. Our architecture guarantees access pattern hiding not only between every client and the server, but also between the clients.

Once we have established the necessary theoretical background and proposed our ORAM constructions that achieve the privacy requirements for a modern PET, we apply our solutions to a contemporary data intensive and privacy sensitive problem. An ideal field to demonstrate the potential of our PETs is that of genomic privacy. In the past years, the rapid developments in genome processing have made it possible to process the human genome and store it digitally. The applications of such a technology will most probably have the potential to eliminate most of today's life threatening diseases. These advancements however do come at a cost which if not taken care of can reverse the technological utopia into a sociological nightmare. The reason for this is exactly the sensitivity of genomic data. Take for example the case where a human genetics laboratory performs a pre-implantation genetic diagnosis for, e.g., autism or Altzheimer's disease and the correlated presence of a specific mutation or combination thereof. Knowing that these diseases are associated with specific parts of the human genome [Seb+07; Dab+10] the mere fact that these certain parts of the genetic data are accessed in the parental genomes already leaks privacy-sensitive information such as the fact that an implantation is to occur, who the parents are, and most importantly that the parents wonder about their own susceptibility to a disease. Ideally, all these facts should be hidden from the (commercial) entity that stores the genomic data.

One would think that since secure computation based solutions are mature enough to handle mid-size problems such as genome processing based on SNPs, they should have already found their way in the industry to some extent. Yet, this is not the case, as most of the research on genomic data today is done on unencrypted data that is 'protected' and considered 'secure', simply because it is kept behind Demilitarised Zones (DMZs). We argue that this is not adequate and secure enough protection for highly sensitive data such as genomic data. In fact, a legal framework that will protect the data is already underway, and thus it is not hard to imagine that in the near future, legislation will be in place that will enforce the adoption of privacy preserving technological solutions.

In this work, we test our solutions by adapting them to the problem of privacy-preserving genomic data. We thus obtain solutions that are *flexible to support future query types*, are applicable to *fully sequenced genomes* and allow to *hide the nature of the performed test*. To this end, we store the sequenced genome at a server in encrypted form using the ORAMs that we developed. In a subsequent query phase the stored

data can be accessed and utilised in a secure computation. By separating secure storage from the query phase, we achieve full flexibility to adapt to future queries. We achieve efficiency by only requesting as input to the secure computation phase those parts of the genomic string that are actually necessary to perform the computation and not the entire genomic sequence – while hiding the queried positions themselves. This enables query times that are sublinear in the length of the genomic sequence. Finally, due to the use of an ORAM, the party storing the genome is fully oblivious on the data accessed during a query; he only sees the *amount* of data accessed, but not its location in the genome.

Thesis Outline

We begin in Chapter 2 by reviewing recent advancements in ORAM solutions and genomic studies thus obtaining an overview of the current state-of-the-art solutions, and motivating the problems dealt with in this thesis. In Chapter 3, we present the necessary mathematical and cryptographic building blocks that will be used in this work. We present homomorphic encryption systems that will be used throughout the course of the thesis, and present Stefanov et al. Path-ORAM [Ste+13a]. Finally, we briefly go through Yao's garbled circuit solution for Secure Two-Party Computation.

Our new ORAM framework is presented in Chapter 4. We introduce our novel and flexible ORAM definition, and define ORAM security by means of a security game inspired by the IND-CPA game for encryption schemes. We then show that our new definition is equivalent to the simulation based security definition of Garg et al. [GMP16], and conclude the chapter by using our new framework in order to describe Path-ORAM [Ste+13a] and prove it secure.

In Chapter 5 we introduce our first ORAM construction, Proxy-ORAM. This solution, allows a client to store encrypted data on a remote server and give partial access to third parties. The latter can access this data without revealing to the server any information except for the amount of accesses performed, while the third party sees the data only in encrypted form.

Chapter 6 deals with the study of Partial Sharing Multi-Client ORAMs, i.e., ORAM Architectures that allow multiple clients to store their data on a server and share between them parts of their data. We provide the necessary theoretical background for such a study by extending the framework introduced in Chapter 4, and present Blurry-ORAM, the first instantiation of a Partial Sharing Multi-Client ORAM.

In Chapter 7, we apply our ORAM architectures to the problem of Genomic Studies. For this, we combine Proxy-ORAM with Yao's Garbled Circuit Solution for Secure Two-Party Computation. This way, we provide a solution that allows a third party to access parts of a client's sequenced genomes in a privacy preserving manner, and then use Yao's Garbled Circuits in order to perform a variety of tests on encrypted data and learn only the end result of the computations. Consequently, we use Blurry-ORAM to store encrypted sequenced genome of multiple clients, and using the partial-sharing attribute of this architecture we can perform Genome Wide Association Studies in a privacy preserving way. We implement both approaches, and the experimental results we present in this chapter.

We conclude this thesis in Chapter 8, with open problems and ideas for future work.

Publications used in this work

This thesis is based on the following publications:

- Nikolaos P. Karvelas, Andreas Peter, Stefan Katzenbeisser, Erik Tews, and Kay Hamacher. *Privacy-Preserving Whole Genome Sequence Processing through Proxy-Aided ORAM*. In: *Proceedings of the 13th Workshop on Privacy in the Electronic Society, WPES 2014*. ACM, 2014.
- Tommaso Gagliardoni, Nikolaos P. Karvelas, and Stefan Katzenbeisser. *ORAMs in a Quantum World*. In: *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, Vol. 10346. Lecture Notes in Computer Science. Springer, 2017.
- Nikolaos P. Karvelas, Andreas Peter, and Stefan Katzenbeisser. *Using Oblivious RAM in Genomic Studies*. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2017 International Workshops, DPM 2017 and CBT 2017*. Vol. 10436. Lecture Notes in Computer Science. Springer, 2017.

Chapter 2

Related Work

Privacy Enhancing Technologies have been a very active field of research that has been studied extensively in the past decades. Moreover, the field of genomic privacy has also received ceaseless attention in the last years, primarily due to the grave importance that protection of genomic data will most probably play in the digital society of the future. Therefore, it would not be an exaggeration to argue that presenting even the majority of the results of the two aforementioned research fields would be a task well outside the scope of this work. We do however give a brief overview of some important results that motivate the constructions and solutions that will be presented later in this work, namely the basic core of ORAM constructions and works on privacy preserving genomic studies.

2.1. Oblivious RAM

Due to its abstract and generic nature, describing an ORAM on a high level can be challenging. For reasons of simplicity, as well as due to its use in this work, we will describe ORAMs in a client-server setting: A client outsources his encrypted private datablocks to a remote server and wants to access his data, revealing to the server nothing more than the amount of blocks stored and the number of accesses performed. With these in mind, we briefly describe in the following, the main ORAM families as they have developed historically. At the same time, we propose a categorization of the various ORAMs in respect to the notions of multi-client and parallelization characteristics.

2.1.1. Basic Constructions

Square-Root Solution

The original solution of Goldreich in [Gol87; GO96] works as follows: The client's N blocks (denoted as 'real') are stored in encrypted manner on the server's so-called 'Permuted Memory', and under a permutation only known to the client. Additionally a smaller buffer able to hold $\sqrt{N}$ blocks, called 'Shelter' is used originally filled up with 'fake' (or 'dummy') blocks[1]. In order to access one datablock,

[1] Each of these blocks is of the same size as a real datablock, is filled with '0', and encrypted under the same semantically secure encryption scheme used to encrypt the real blocks

the client first scans the entire 'Shelter'. If the datablock was found in the shelter, then the client gets a random datablock from the 'Permuted Memory'. Otherwise, the client gets the correct datablock from the 'Permuted Memory'. After the datablock has been read or written, the client puts it back to the 'Shelter', and re-randomises the entire Shelter. After $\sqrt{N}$ access, the Shelter will be filled with real datablocks, which in an interactive step are then obliviously emptied to the Permuted Memory, by using e.g. sorting networks, a process which is called a 'reshuffle'. This process results in an *amortized* computation and communication cost of $O(\sqrt{N}\log^2 N)$.

Hierarchical ORAMs

In order to avoid the high communication costs of the Square-Root solution's reshuffle step, Goldreich and Ostrovsky introduced in [GO96] also the notion of Hierarchical ORAMs. There, the client's N real datablocks are stored on the server, along with an equal amount of 'fake' datablocks, in a pyramid-like structure of $\log N$ levels. Each level is represented as a hash table, consisting of a level dependent number of fixed sized buckets, to which datablocks are assigned. In order to retrieve one datablock from the ORAM, the client downloads and decrypts one bucket from every level, as indicated by the level's hash table. In one of those buckets the client is guaranteed to find the datablock he was querying for, while for the server the accesses look totally random. After the client has finished his query, he writes the datablock back into the first level's bucket: If the query was a read operation, the client writes back the datablock re-encrypted, while in case of a write operation, he writes back an encryption of the updated datablock. Once the buckets of a level are filled, a 'reshuffle' operation takes place: The level's buckets are obliviously emptied to the next level, in such a way that any correlation between the datablocks accessed on the two levels is destroyed. This results in an amortised access overhead of $O(\log^3 N)$, while the client needs to accommodate constant local storage.

In the years that followed, the hierarchical solution met many optimisations, and many variants of the original scheme were proposed. Most notably, the solution of Pinkas and Reinman [PR10] used Cuckoo hashing and gained a logarithmic factor in the communication complexity. However, it was shown to suffer from a privacy leakage, which was not further addressed. Subsequent works [WSC08; WST12] replaced the hash table with a Bloom filter. A query for a datablock runs in a similar way as before, but instead of looking at the hash table in order to see if the datablock is on the level and downloading the corresponding bucket, the client now consults the corresponding Bloom filter and downloads either the real datablock (if it was found on that level) or a fake one, thus saving a logarithmic communication factor per query. The reshuffle is also simplified: after two levels are merged, one only needs to re-randomise the elements and build a new Bloom filter.

Tree-based ORAMs

The ORAM landscape changed rapidly with the introduction of tree-based ORAMs. The first such scheme was proposed by Shi et al. [Shi+11]. The core idea is to map each real datablock to a leaf of a binary tree data structure. Reading a datablock is done, by scanning the path to the leaf where the datablock is mapped to, and while doing this, rearranging all datablocks found in the path so that as

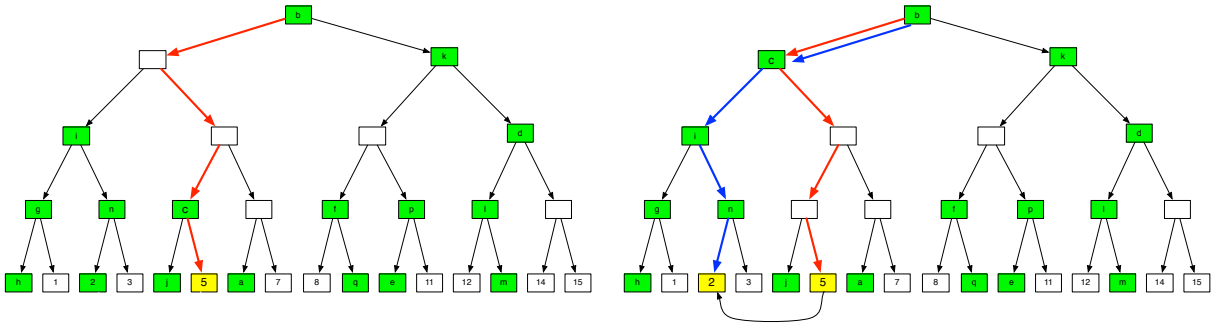


Figure 2.1: Simplified example of Path-ORAM access: The datablock 'c' is found along the path to leaf number '5' (left); then datablock 'c' is remapped to leaf number '2', and moved to the common ancestor of paths to leaves '2' and '5', which is closer to the leaf level (right).

many datablocks as possible are pushed closer to the tree's leaf level, in an operation called 'eviction'. This approach achieved a *worst-case* communication overhead of $O(\log^3 N)$ while maintaining *constant* client storage.

The idea behind Shi et al. construction, was further explored by Stefanov et al. [Ste+13a] where the eviction procedure was simplified, resulting in the Path-ORAM construction which constitutes today's state-of-the-art ORAM architecture. Again here, each datablock is mapped randomly to a leaf of the tree, an after the datablock has been found, its position is randomly remapped. This time however, the datablock is placed on the common ancestor of the paths leading to the old and the new leaves, and that is closer to the leaf level (a graphical representation of this procedure can be found in Figure 2.1). This scheme achieves a communication overhead of $O(\log^2 N)$, with the client needing $O(\log N)$ local space. In fact, it is Path-ORAM's practical efficiency that made it the perfect candidate in the construction of the secure processor of Maas et al. [Maa+13].

Hybrid and other ORAMs

From the above discussion, it becomes clear that the efficiency of the various ORAM constructions depends on a multitude of parameters. Thus, combining ideas from various ORAM schemes, in a series of works, Stefanov et al. [SS13a; SSS12; SS13b] proposed schemata, where the client's data is distributed among a number of servers in a partitioning-like paradigm. Each of these partitions stores the client's datablocks using the square-root or the hierarchical solution. Assuming that the client has less storage restrictions than the ones originally imposed by Goldreich and Ostrovsky, these hybrid architectures result in practically efficient solutions, since the database is split into several smaller ones, which can be more efficiently and in parallel accessible.

In a similar manner, however not using the partitioning-framework, Goodrich et al. [Goo+12a] devise an ORAM that achieves constant amortised communication costs however by burdening the client with local storage of size $O(N^{1/c})$, for some $c \geq 2$. Finally, Dautrich and Stefanov [JSS14] proposed an ORAM that exploits the fact that clients tend to perform many accesses in a short time after long times

of idleness, i.e., 'bursts'. This results in a very efficient ORAM however again under heavy demands on the client's local storage. [WCS15].

2.1.2. Multi-Client ORAMs

ORAMs arose from the need to protect against software piracy, and as such, ORAM research was mostly concerned about the model of a processor reading and writing to its memory module (a model very similar to having a client accessing his private data to a remote server). For most modern systems or scenarios however, such a setting is either over-simplifying or even obsolete, and thus the notion of multi-client ORAMs came to light where a dataset that is accessible by a number of clients is considered. In such a setting, the mere existence of a multitude of clients gives rise to many interesting privacy questions, which have been dealt with in a number of works, and which we will briefly recall in the following.

The first and main question regarding multi-client ORAMs is if they exist in the first place, i.e., if an ORAM construction can be built, which guarantees that different clients that have access to the ORAM, can share between them the ORAM state which gets updated after every access. A positive answer to this question came from the construction of Goodrich et al. [Goo+12b], which is based on the hierarchical solution. Using the Cuckoo hashing technique, the authors showed that the ORAM's state could be communicated between the clients in a private way, yielding a so-called *stateless* ORAM. However, this came at the expense of the clients' memory, who for N outsourced datablocks had to dedicate $O(N^{\nu})$ memory for some fixed constant $\nu > 0$.

In a parallel and independent work, Williams et al. [WST12] solved the same problem by introducing the notion of a *period-based stateless* ORAM, by employing a server-stored and encrypted 'log' of accesses which is shared and consulted by the clients during every access. The main purpose of this log was to allow *parallel* access during one period. The solution was again based on the hierarchical ORAM, this time however in its more efficient Bloom filter variant from [WSC08], which allowed for the first instance of a privacy preserving parallel filesystem.

Another interesting problem arising once multiple clients are allowed to access the same ORAM is client *anonymity*, which can be seen as the inability of the server and other clients to link a client to a specific operation on a datablock. This problem was addressed by Backes et al. in [Bac+16], where the proposed solution employs two non-colluding servers in a hierarchical based ORAM and achieves security against malicious adversaries.

The last and for this thesis most relevant security question regarding multi-client ORAMs, is the information leakage arising when multiple clients share only *parts* of their data with each other. Pioneering the idea of an ORAM that allows such a partial sharing of data in the presence of multiple clients, Franz et al. [Fra+12] introduced an ORAM that allows *delegation* of rights from a data owner to other clients. The main idea is that every datablock stored on the server is encrypted with a different key, which the data owner hands over to the client, in order to delegate read or write access rights to that client for the particular datablock. Using this approach however, the data owner has to be constantly online in order to

perform this costly operation since the client does not know all the keys in order to perform an ORAM reshuffle. Being based on the square-root solution, this solution further suffers from high communication costs not only during the reshuffle operation but also during reading or writing datablocks.

In a more recent work Maffei et al. [Maf+15] addressed the above problem in a more holistic way: the authors assumed the existence of multiple clients and proposed a solution that provides strong security guarantees against malicious adversaries. First, the access patterns are hidden from the server. Secondly, the integrity of the data is protected against a malicious server and against malicious clients. Most importantly, the clients are guaranteed to be able to read only the entries that they have access to. Being based on the highly efficient Path-ORAM construction [Ste+13a], this solutions enjoys also high efficiency, alongside its strong security guarantees. However, this solution does not consider the potential information leakage that can be inferred by datablocks shared between multiple clients. In particular, even when the clients can only access the datablocks for which they have permissions, still by merely observing the changes in the recursively stored position map, they can see *exactly* which datablocks have been accessed by other clients. In a similar manner, the existence of the a stash which is shared by all the clients, can learn important information such as if in a particular bucket a real or a fake datablock is stored.

2.2. Genomic Data

We begin by recalling some basic facts on genomic sequencing that will help us better understand the privacy problems associated with it. The human genome is a set of nucleotides structured in two complementary polymer chains, with the four nucleotide bases of the polymer chain being adenine, cytosine, guanine and thymine, represented by A, C, G, T respectively. In order to determine the exact *raw* sequence of a human DNA, various techniques have been developed in the past years, including illumina dye sequencing, pyrosequencing or single molecule real time sequencing. The raw sequence is then further analysed by being aligned to the so-called *reference genome* which is considered to be a typical representation of our specie's genome. The result of this procedure is a set of approximately 3.2 billion characters, stored in the Sequence Alignment Map (SAM) file. Out of this set of characters, only around 0.5% differs between various individuals and seems to be significant in uniquely identifying a specific individual. Thus, the way that a human genome is stored, is by identifying these positions (loci) which form the most common DNA variation and are called SNP. Once a person's DNA has been sequenced and analysed, it can be stored and be used for further tests that range from simple paternity and ancestry tests, to Short Tandem Repeat (STR) analyses (i.e., the study of repetitive DNA patterns associated with higher mutation rate than other DNA parts) and to Genome Wide Association Studies, the latter being studies ran among multitudes of different genome sequences, and whose aim is to determine (among other) which parts of the DNA are responsible for particular diseases.

DNA can be easily seen as a person's most private and sensitive data, as it does not contain only information about the particular individual, but also about the individual's predecessors, relatives and

descendants. Thus, early on after the breakthrough results in Biology that allowed the digital sequencing, storing and processing of human DNA, cryptography based solutions were developed that allowed the enhancement of security and privacy for this highly sensitive set of data. Early works focused on processing short DNA fragments and were tailored towards simple queries: for example, [TKC07] showed how to run queries in the form of finite state machines on DNA sequences, which was subsequently improved in [BA10; Fri09; WR13]. Unfortunately due to their computational complexity, these approaches are only applicable to small fragments of DNA strings and not to the entire genome. Further, they are tailored towards very specific queries, which can be represented as finite state machines.

Similarly, works that allow to search for patterns in strings, such as [GHS10; HT10; KM10], can be applied to the problem domain. Nevertheless, again the solutions are limited to very specific search queries and incur an overhead that is linear in both the size of the genome and the search pattern. Solutions tailored to perform specific forensic tests efficiently have been proposed as well [Bru+08; KM10]. Only a couple of works targeted efficient private queries on fully sequenced genomes: [Bal+11a] focused on targeted tests for paternity, personalized medicine applications and genetic compatibility; technically, they employ secure set intersection protocols, which are extremely efficient but only work for simple queries. Finally, [ARH13] proposed an architecture based on homomorphic encryption which is flexible with respect to the query, but leaks the type of test performed to the involved parties. Still, they lack the capacity to learn and answer important 'fuzzy' (statistical) questions – namely to act in concert with biostatistics. Note, that all of these approaches apply to digitally represented and sequenced genomic data; they can be amended by privacy-preserving genomic sequencing techniques [Che+12].

In [Ayd+13] a solution was proposed, that hides the identities of clients, as well as the nature of the tests performed when the clients store their SAM. For every query the server has to go through the whole database, thus the solution suffers from high computational costs. In [Hum+15] de-anonymisation attacks were examined, that showed that due to phenotypical correlations, user privacy is severely compromised if no cryptographic techniques are employed in order to order to protect the data.

Baldi et al. [Bal+11b] addressed the problem of privately searching in a whole sequenced human genome, by means of Private Set Intersection, a primitive which has been further optimized in the past years [PSZ14; Kis+17]. However, this solution can deal only with a specific type of query in the genome, and lacks the general applicability that we want to achieve.

Recently, Demmler et al. [Dem+17] proposed an ingenious solution of performing whole genome variant queries into large databases that store the genomic data of multiple clients in a privacy preserving way. The core idea of their construction is to 'secret-share' databases that store the genomic data of multiple clients and run the Goldreich-Micali-Wigderson [GMW87] for secure multi-party computation. Although this scheme is very efficient and protects the user identity and the type of the queries performed, it does not allow any database updates. We argue that this is an important feature of any solution that promises privacy of genomic databases, since it is known that genome changes throughout the life of a client, and this will have to be accounted for, for any reasonable studies of biostatistics.

Chapter 3

Preliminaries

Once a cryptographic primitive or protocol has been introduced, arguing about their security guarantees is not a trivial task. Thus, it is not a surprise that it took the research community almost 40 years in order to make the transition from the 'cat-and-mouse' security arguments of the early days of cryptology (where a primitive was considered secure, as long as none of the known attacks against it was successful) to the modern age of *provable security*. Today, security proofs are done by means of *mathematical reductions*: Assuming that a specific problem P is hard to solve for any powerful machine while the security of the cryptographic primitive C under consideration can be easily broken, we construct an efficient algorithm that turns an instance of C into an instance of P, and hence by efficiently solving C, the algorithm solves P which in turn yields a contradiction. Yet the original 'cat-and-mouse' approach has not been entirely abandoned, since in many cases the attacks considered are not simply the ones developed in the past years, but include all the attempts made by mathematicians of all known history to solve particular problems. One such example is the factoring problem, for which no efficient algorithm is known to exist since Euclid's first proof of the *Fundamental Theorem of Arithmetic* [Euc56]. Nevertheless, today's provable security is the only way we can reasonably argue about a scheme's security. Most importantly, probably the best indicator for the success of provable security is the fact that once a scheme is proven secure, the real world attackers mainly target the error-prone implementations of the scheme (e.g. [Adr+15]) and almost never the scheme's mathematical building blocks.

With the above thoughts in mind, in this chapter we will go through the necessary notions that will help us not only build complex cryptographic protocols, but also show that these protocols are secure against polynomial time probabilistic adversaries.

3.1. Basic Tools and Cryptographic Primitives

Let $\mathbb{N}$ be the set of positive integers. For a finite set X, the notation $x \xleftarrow{\$} X$ indicates that x is selected uniformly and at random from X. For a probability distribution S, the notation $x \leftarrow S$ indicates that x is sampled according to S. We say that a function $f : \mathbb{N} \to R$ is *polynomially bounded* if and only if there exists a polynomial p and an $n_0 \in \mathbb{N}$ such that: for every $n \geq n_0$ it holds that $f(n) \leq p(n)$, in which

case we will just write $f = \text{poly}n$. We say that a function $\epsilon : \mathbb{N} \to R$ is *negligible*, if and only if for every polynomial p, there exists an $n_0 \in \mathbb{N}$ such that $\epsilon(n) \leq \frac{1}{p(n)}$ for every $n \geq n_0$; in this case we will just write $\epsilon = \text{negl}(n)$.

In the rest of this work $\lambda \in \mathbb{N}$ will denote the security parameter, and $\perp$ will be used as a special symbol denoting an empty bitstring of finite length. PPT stands for 'probabilistic polynomial time', and by 'algorithm' we will mean a uniform family of circuits; therefore 'Probabilistic Polynomial Time (PPT) algorithm' will stand for 'uniform family of poly-sized Boolean circuits with randomness'. Given a security *game*, or *experiment*, the output 1 will denote success (winning condition), while 0 will denote failure (loss condition).

Definition 1 (Encryption Scheme)**.**
An encryption scheme is a triple of probabilistic polynomial-time algorithms $(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ *that satisfies the following conditions:*

1. *On input* 1^λ*, the key generation algorithm* KeyGen *outputs a pair of bitstrings* (e, d)*.*
2. *For every pair* (e, d) *in the range of* $\texttt{KeyGen}(1^\lambda)$*, and for every* $m \in \{0,1\}^*$*, the encryption and decryption algorithms,* Enc *and* Dec *respectively satisfy the following:*

$$\text{Prob}[\texttt{Dec}(d, \texttt{Enc}(e, m)) = m] = 1.$$

Definition 2 (Public-Key and Secret-Key Encryption Schemes)**.**
Let $\mathcal{E} = (\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ *be an encryption scheme as of Definition 1, that on input* 1^λ *outputs a pair of keys* (e, d)*. We say that* $\mathcal{E}$ *is a private-key encryption scheme, if for every encryption-decryption key pair* (e, d) *we have that* $e = d$*, and we say that* $\mathcal{E}$ *is a public-key encryption scheme otherwise.*

Definition 3 ($\mathsf{Game}^{\texttt{IND-CPA}}_{\texttt{A,Enc}}(\lambda)$)**.**
Let $\mathcal{E} = (\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ *be an encryption scheme,* λ *a security parameter and* A *an adversary for the encryption scheme. The computational indistinguishability of ciphertexts game under adaptive chosen query attack* $\mathsf{Game}^{\textit{IND-CPA}}_{\texttt{A,Enc}}(\lambda)$ *proceeds as follows:*

1. $(e, d) \leftarrow \texttt{KeyGen}(\lambda)$*;*
2. *First CPA learning phase: for* $i = 1, \ldots, q_1 \in \mathbb{N}$*,* A *repeats (adaptively) the following:*
   a) A *chooses a message* m_i*;*
   b) C *encrypts* m_i *using* Enc*;*
   c) A *receives* $\texttt{Enc}(m_i)$*;*
3. *Challenge phase:* A *chooses two messages* m^0 *and* m^1*;*
4. C *flips a random secret bit* $b \xleftarrow{\$} \{0,1\}$ *and encrypts* m^b*;*
5. A *receives* $\texttt{Enc}(m^b)$*;*
6. *Second CPA learning phase: for* $j = 1, \ldots, q_2 \in \mathbb{N}$*,* A *repeats (adaptively) the following:*

a) $\mathtt{A}$ *chooses a message* m_j*;*

b) $\mathtt{C}$ *encrypts* m_j *using* $\mathtt{Enc}$*;*

c) $\mathtt{A}$ *receives* $\mathtt{Enc}(m_j)$*;*

7. $\mathtt{A}$ *outputs a bit* b'.

$\mathtt{A}$ *wins the game iff* $b = b'$.

Definition 4 (IND-CPA)**.**
An encryption scheme $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ *has indistinguishable ciphertexts under chosen plaintext attack (or, it is IND-CPA-secure) if and only if for any PPT algorithm* $\mathtt{A}$ *with oracle access to* $\mathtt{Enc}$ *there exists a negligible function negl such that:*

$$\left|\mathrm{Prob}\left[\mathsf{Game}_{\mathtt{A},\mathtt{Enc}}^{IND\text{-}CPA}(n) = 1\right] - \frac{1}{2}\right| \leq negl(\lambda).$$

Intuitively, an encryption scheme is IND-CPA-secure if no computationally bounded adversary can reliably distinguish between the encryption of two plaintexts of his choice with probability non-negligibly better than guessing, even if allowed to adaptively learn polynomially many other encryptions.

In our constructions, we will be considering many clients storing their encrypted data on a remote server. One of the necessary requirements for user privacy in these settings will be that data encrypted under different keys should not reveal anything about the keys. This property is captured under the notion of *key indistinguishability*, which was introduced by Bellare et al. [Bel+01a] by means of a security game, where an adversary is given two public keys generated by an encryption scheme's $\mathtt{KeyGen}$ function, and an encryption of a message under one of those two keys, chosen at random. The encryption scheme is said to provide key indistinguishability if no adversary can correctly guesses the key under which the message has been encrypted. In more the detail, the key indistinguishability property is defined as follows.

Definition 5 (IK-CPA)**.**
Let $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ *be an encryption scheme. Let* $b \in \{0,1\}$ *and* $k \in \mathbb{N}$*. Let* $\mathtt{A}$ *be an adversary that runs in two stages. The IK-CPA-b game for a security parameter* λ*,* $\mathsf{Game}_{\mathcal{E},\mathtt{A}}^{\text{ik-cpa-b}}(\lambda)$ *is as follows:*

1. $(\mathtt{pk}_0, \mathtt{sk}_0) \xleftarrow{R} \mathtt{KeyGen}(\lambda)$*;* $(\mathtt{pk}_1, \mathtt{sk}_1) \xleftarrow{R} \mathtt{KeyGen}(\lambda)$*;*
2. $(x, s) \leftarrow \mathtt{A}(\mathrm{find}, \mathtt{pk}_0, \mathtt{pk}_1)$
3. $y \leftarrow \mathtt{Enc}_{\mathtt{pk}_b}(x)$*;*
4. $d \leftarrow \mathtt{A}(\mathrm{guess}, y, s)$
5. *Return* d

The advantage of $\mathtt{A}$ *is defined as*

$$\mathtt{Adv}^{\text{ik-cpa-b}}_{\mathcal{E},\mathtt{A}}(\lambda) = \text{Prob}\Big[\mathsf{Game}^{\text{ik-cpa-1}}_{\mathcal{E},\mathtt{A}}(\lambda) = 1\Big] - \text{Prob}\Big[\mathsf{Game}^{\text{ik-cpa-0}}_{\mathcal{E},\mathtt{A}}(\lambda) = 1\Big].$$

The scheme $\mathcal{E}$ *is said to be IK-CPA secure if* $\mathtt{Adv}^{\text{ik-cpa-b}}_{\mathcal{E},\mathtt{A}}(\lambda)$ *is negligible in* λ*, for any PPT adversary* $\mathtt{A}$*,*

The way a series of numbers can be proven to be pseudorandom is critical in most modern cryptographic solutions. We thus recall the definition of a pseudorandom number generator in the following.

Definition 6 (PRNG)**.**
Let ℓ *be a polynomial such that* $\ell(n) \geq n+1 \ \forall n \in \mathbb{N}$*. A pseudorandom number generator, or PRNG with expansion factor* $\ell(.)$ *is a deterministic polynomial-time algorithm* $\mathcal{G} = \mathcal{G}_\ell$ *such that given as input a bitstring* s *of length* n*, (the seed), outputs a bitstring* $\mathcal{G}(s)$ *of length* $\ell(n)$*; and for any PPT algorithm* D*:* $\Big|\text{Prob}\big[D(r)=1\big] - \text{Prob}\big[D(\mathcal{G}(s))=1\big]\Big| \leq negl(n)$*, where* $r \xleftarrow{\$} \{0,1\}^{\ell(n)}, s \xleftarrow{\$} \{0,1\}^n$*, and the probabilities are over the choice of* r *and* s*, and the randomness of* D*.*

Bloom filters [Blo70]

A Bloom Filter is a randomised data structure storing a set of elements $S = (s_1, \ldots, s_n)$ and which returns true with probability 1 if an element is a member of S. If however, $s \notin S$ the Bloom filter returns false with probability p and true with probability $1-p$. A Bloom filter is usually implemented as an array B of m bits, initialised to the 0-string, together with a total number of ξ hash functions $\{h_j\}_{j=1}^{\xi} : \{0,1\}^* \to [m]$, such that for every element $s \in S$ it holds that $B[h_j(s)] = 1$ for all $j = 1, \ldots, \xi$. Testing if an element s is in the Bloom filter amounts to verifying that $B[h_j(s)] = 1$ for every $j = 1 \ldots, \xi$. Although estimating the Bloom filter false positive ratio seems to be a trivial task at first, thorough analyses in the past years [CRJ10; Bos+08], have shown that this is not the case, and one must be careful when arguing about the probability of a false positive when employing these data structures.

3.2. Finite Group Based Cryptography

For a natural number p, the set of congruence classes relatively prime to the integers modulus p is arguably the easiest example of a multiplicative, finite Abelian group, denoted as $(\mathbb{Z}_p^*, \cdot)$.

For finite Abelian groups, the decisional Diffie Hellman problem can be reduced to the Discrete Logarithm problem (for a group $G = \langle g \rangle$ of order m and an element $y \in G$, find y, given g^y), and is defined as follows:

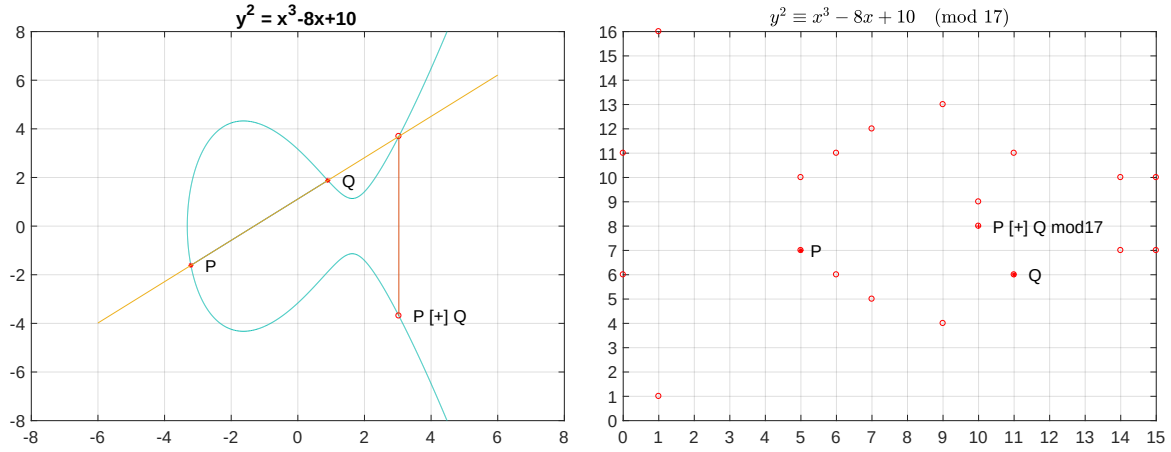


Figure 3.1: Continuous (left) and discrete (right) Elliptic curve, and their respective addition law (here noted with '[+]')

Definition 7 (Decisional Diffie Hellman (DDH) Problem)**.**
Let $G = \langle g \rangle$ be a group of order m. Given a tuple (g^a, g^b, g^c) for some $a \xleftarrow{\$} \mathbb{Z}_m$, $b \xleftarrow{\$} \mathbb{Z}_m$, and $c \xleftarrow{\$} \mathbb{Z}_m$ or $c = ab$, decide if $c \xleftarrow{\$} \mathbb{Z}_m$ or $c = ab$.

Another interesting, yet more involved example of finite Abelian groups is that of Elliptic Curves. For a formal treatment of Elliptic Curves we refer to [Sil16], while here we only restrain ourselves to a high level and intuitive approach that will help us later describe the ElGamal encryption scheme [Gam85] on this algebraic structure.

An elliptic curve over a finite field[1] $\mathbb{K}$ which is not of characteristic 2 or 3 (i.e., the smallest n such that $n \cdot 1_{\mathbb{K}} = 0_{\mathbb{K}}$) is the set of solutions of the generalised cubic equation $Y^2 = X^3 + aX + b$, with $a, b \in \mathbb{K}$ and a *point at infinity* set to the identity $0_{\mathbb{E}}$. The group operation for two points of an elliptic curve $\mathbb{E}$, P and Q is another point Z of the curve, viewed as the point of intersection of the line passing though P and Q, and the curve. For any points it must hold that $P + Q + Z = 0_{\mathbb{E}}$, and for the two points $P = (x_p, y_p)$, $Q = (x_q, y_q)$ we can calculate $Z = (x_z, y_z)$ as $x_z = \mu^2 - x_q - x_p$ and $y_z = y_p + \mu(x_z - x_p)$ with the slope μ between X and Y being $(y_q - y_p)/(x_q - x_p)$ if $P \neq Q$ and $(3x_p^2 + a)/(2y_p)$ if $P = Q$. For an example of a graphical representation of this operation in the continuous and discrete case of an elliptic curve we refer to Figure 3.1.

3.2.1. Homomorphic Encryption

For two groups (G_1, op_1) and (G_2, op_2), a group homomorphism is defined as a function $\phi : G_1 \to G_2$, such that for all $x, y \in G_1$ it holds that $\phi(\mathsf{op}_1(x, y)) = \mathsf{op}_2(\phi(x), \phi(y))$. As we shall see shortly, there exist various encryption schemes whose encryption function $\mathtt{Enc}$ actually are homomorphisms between the plain- and ciphertext domains, and thus allow manipulation of the underlying plaintexts while

[1] a field is an algebraic structure on a set $\mathcal{K}$, equipped with two operations $\mathsf{op}_1, \mathsf{op}_2$, such that $(\mathcal{K}, \mathsf{op}_1)$ is an Abelian group with identity $0_{\mathcal{K}}$ and $(\mathcal{K} \setminus \{0_{\mathcal{K}}\}, \mathsf{op}_2)$ is an Abelian group with identity $1_{\mathcal{K}}$, and the distributive law holds.

operating on ciphertexts. In fact, we will make heavy use of these encryption schemes and their respective homomorphic properties throughout this work, since these properties will allow us to perform a variety of operations on encrypted data, such as re-randomising encrypted data with or without knowledge of the encryption key, changing the secret key without firstly decrypting the ciphertext (a process called 'proxy re-encryption') or computing on encrypted data.

ElGamal Encryption Scheme [Gam85]

The ElGamal encryption scheme is a public key homomorphic encryption scheme that has revolutionised modern cryptography. When instantiated with the multiplicative group of integers modulo n, it works as follows: For a security parameter λ, $\texttt{KeyGen}(\lambda)$ chooses a safe prime p, finds a generator for $\mathbb{Z}_p^*$, selects a random group element a which will server as the secret key $\texttt{sk}$, and sets as public parameters the set (p, g, g^a) where g is a group generator and g^a is the public key $\texttt{pk}$. For a message $m \in \mathbb{Z}_p^*$, the encryption algorithm $\texttt{Enc}_{\texttt{pk}}(m)$ chooses a random element $y \xleftarrow{\$} \mathbb{Z}_p^*$ and outputs $(A, B) = (g^r, m \cdot g^{a \cdot r})$. The decryption algorithm $\texttt{Dec}_{\texttt{sk}}(\texttt{Enc}_{\texttt{pk}}(m))$ outputs the plaintext as $m = B \cdot A^{-a}$.

When instantiated using Elliptic Curves, the ElGamal encryption scheme works as follows: For a security parameter λ, $\texttt{KeyGen}(\lambda)$ sets an Elliptic Curve $\mathbb{E}$ over a field $\mathbb{F}_q$ for a prime q, selects a point P of order N, selects a random element $a \in \{1, \ldots, N-1\}$ which will server as the secret key $\texttt{sk}$, and sets as public parameters the set $(\mathbb{E}, P, xP)$ where xP is the public key $\texttt{pk}$. Further a publicly known, one way and easily invertible function $f : m \mapsto P_m$ is used, that maps a random message m to a unique point of the elliptic curve. For a message P_m, the encryption algorithm $\texttt{Enc}_{\texttt{pk}}(P_m)$ chooses a random element $y \xleftarrow{\$} \{1, \ldots, N-1\}$ and outputs $(A, B) = (yP, yxP + P_m)$. The decryption algorithm $\texttt{Dec}_{\texttt{sk}}(\texttt{Enc}_{\texttt{pk}}(P_m))$ outputs the plaintext as $m = B - aA$.

The ElGamal encryption scheme is semantically secure under the DDH assumption and is multiplicatively homomorphic, i.e., $\texttt{Dec}_{sk}(\texttt{Enc}_{\texttt{pk}}(m) \cdot \texttt{Enc}_{pk}(m')) = m \cdot m'$.

Bresson-Catalano-Pointcheval Encryption [BCP03]

For a security parameter λ, $\texttt{KeyGen}(\lambda)$ chooses a λ-bit safe-prime modulus $N = pq$ (i.e. $p = 2p' + 1$, $q = 2q' + 1$ for two distinct primes p', q') and picks a random element $g \in \mathbb{Z}_{N^2}^*$ of order $pp'qq'$, such that $g^{p'q'} \mod N^2 = 1 + kN$, for $k \in [1, N-1]$. The plaintext space is $\mathbb{Z}_N$ and the algorithm outputs the public parameters (N, k, g). The key generation algorithm $\texttt{KeyGen}(N, k, g)$ outputs a random element $a \in \mathbb{Z}_{N^2}^*$ as *secret* key and the element $h = g^a \mod N^2$ as *public* key. The encryption algorithm $\texttt{Enc}_{\texttt{pk}}(m)$ picks a random pad $r \in \mathbb{Z}_{N^2}$ and outputs the ciphertext

$$(A, B) = (g^r \mod N^2, h^r(1 + mN) \mod N^2).$$

The decryption algorithm $\texttt{Dec}_{\texttt{sk}}(A, B)$ outputs the plaintext as

$$m = \left(B(A^a)^{-1} - 1 \mod N^2\right)^{-1} \mod N.$$

The Bresson-Catalano-Pointcheval (BCP) encryption scheme is semantically secure under the DDH assumption and is additively homomorphic, i.e., $\mathtt{Dec}_{\mathtt{sk}}(\mathtt{Enc}_{\mathtt{pk}}(m) \cdot \mathtt{Enc}_{\mathtt{pk}}(m')) = m + m'$.

ElGamal Proxy Re-Encryption [BBS98]

Proxy reencryption allows a semi-honest proxy to transform a ciphertext, generated by a party A with her public key, into an encryption under the public key of another party B. The proxy can do so by means of a *re-encryption key for B* given by *A*. Ivan and Dodis [ID03] showed that ElGamal-like schemes (such as the BCP scheme) are in fact proxy re-encryption schemes: Let $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ denote the ElGamal encryption scheme and let (c_1, c_2) be an ElGamal ciphertext with $c_1 = g^r$ and $c_2 = mg^{ra}$ of a message m under the public key g^a for some cyclic group generator g, secret key a, and random exponent r. Consider a "secret sharing" of the key $a = x_1 + x_2$ into two random exponents x_1 and x_2. Then, x_1 becomes the "re-encryption key" and x_2 the new decryption key: Given an encryption (c_1, c_2) under the public key g^a, $(c_1, c_2 \cdot c_1^{-x_1})$ yields an encryption under g^{x_2}.

A Homomorphic Hash Function

Once we have established the existence of homomorphic encryption schemes, we ask ourselves, if there also exist hash functions that have the homomorphic property in the sense that operating on the hash values of two different messages, translates into operating on the messages themselves. Indeed the Chaum van Heisjt Pfitzmann hash function (CvHP) hash function [CHP92], is such a homomorphich hash function and works as follows: Given two primes p and q such that $p = 2q+1$, two elements α and β, $\alpha \neq \beta$ of order q and such that the Discrete Logarithm problem in the group $\langle\alpha\rangle$ generated by α is difficult, the message $m \in \mathbb{Z}_p^*$ is 'split' into m_1 and m_2 ($m_1, m_2 \in \mathbb{Z}_q^*$) and the hash function $h : \mathbb{Z}_q^* \times \mathbb{Z}_q^* \mapsto \mathbb{Z}_p^*$ is computed on m as $h(m_1, m_2) = \alpha^{m_1}\beta^{m_2}$. For another message $m' \in \mathbb{Z}_p^*$, split into m_3 and m_4 ($m_3, m_4 \in \mathbb{Z}_q^*$), the hash function's homomorphic property allows us to compute the hash of the messages' addition by computing $h(m_1 + m_3, m_2 + m_4) = \alpha^{m_1+m_3}\beta^{m_2+m_4}$. Collision resistance follows from the Discrete Logarithm problem.

3.3. Basic Cryptographic Protocols

3.3.1. Path-ORAM

As we have seen in Chapter 2, in the past years a multitude of ORAM constructions has been proposed. Out of the various tree-based ORAM proposed, the Path-ORAM construction of Stefanov et al. [Ste+13a]

is one that has revolutionised the ORAM research. We will make heavy use of this ORAM construction in this work, and so we briefly describe it in the following.

Using a semantically secure encryption scheme $\mathcal{E}$, in Path-ORAM a client stores N encrypted datablocks (referred to as 'real' datablocks), each of B bits in a binary tree structure of N leaves (and thus $\lceil \log_2 N \rceil$ levels). Each node is capable of holding a total of Z datablocks, and typically for an ORAM, so-called 'fake' datablocks are stored in each node, in case the node is not filled up with real datablocks. Each fake datablock is an encryption of padded '0's that have the same length as real datablocks. Since each tree node can hold up to Z datablocks, for N real datablocks, $Z(2N-1)-N$ fake datablocks are stored in the structure. Each real datablock is mapped to a leaf of the tree, and this mapping is stored in a private structure called *'position map'* on the client side[2]. Whenever the client wants to read (or write) one of his datablocks, say block_i, he downloads the whole path (and all the datablocks found along it) from the root node to the respective leaf. Path-ORAM soundness guarantees that the client will find the desired datablock in one of the nodes along this path, with high probability. Once the path has been downloaded, the client chooses randomly a new leaf, re-maps the retrieved datablock to this leaf, and places the datablock in the node closest to the leaf, which is the common ancestor of the retrieved datablock's previous and new mappings, if there is enough space in its buckets. Otherwise the datablock is moved to higher and higher levels (where the highest level is the one holding the root node), until a node with enough space is found. It can happen that the only common ancestor of the two leaves is the root node – in fact, this event occurs with probability $1/2$ during every remapping. Thus, the root may quickly get filled up with datablocks, in which case a small auxiliary storage called a *'stash'* is used to store the datablock instead, which is shown to grow only logarithmically (in the amount of the datablocks stored in the tree). We stress here that the size of the stash affects the soundness and *not* the security of Path-ORAM, since overflowed datablocks would have to be discarded, and thus never be found again. Furthermore, for every real datablock replaced in the path's node, a fake datablock is put in its position and all datablocks in every accessed node are re-randomised. The datablocks found stored in the stash from previous accesses are also examined if they can be evicted from the stash and placed in the downloaded path. The resulting path is then uploaded to the server while the stash (due to its small size) is stored directly on the client.

3.3.2. Oblivious Transfer

An oblivious transfer protocol (OT) is a protocol in which a sender transfers one of multiple messages to a receiver, but it remains oblivious as to which piece has been transferred. At the same time, the receiver can select the message that he wants to retrieve. Here, we will only use 1-out-of-2 OTs, where the sender inputs two l-bit strings m_0, m_1 and the receiver inputs a bit $c \in \{0,1\}$. At the end of the protocol, the receiver obliviously receives m_c such that neither the sender learns the choice c, nor the receiver learns anything about the other message m_{1-c}. OT protocols require asymmetric cryptography

[2] This requires private storage linear in the amount of datablocks. However, as pointed out in [Shi+11], this structure can be recursively stored in smaller Path-ORAMs until a small size is attained.

and were assumed to be very costly in the past. However, in 2003 Ishai et al. [Ish+03] presented the idea of *OT Extension*, which uses a small number of OTs (called the base OTs) in order to obtain a large number of OTs by use of cheap symmetric cryptographic operations. This way, the computational costs of OTs for most interesting applications of Secure Multi-Party Computation (MPC) are significantly reduced.

3.3.3. Secure Two-Party Computation with Yao's Garbled Circuits

Yao's garbled circuits protocol, proposed in the 1980s [Yao86], is the most well-known secure two-party computation protocol, secure in the semi-honest model. The protocol is run between two parties P_A, P_B and operates on functionality descriptions in form of Boolean circuits over binary Boolean gates. To securely evaluate a functionality $f(x, y)$ over their private inputs x and y, both parties agree on a circuit $C_f(x, y)$, which can be seen as the machine code for the protocol.

During protocol execution, one party becomes the circuit generator (the garbling party), the other the circuit evaluator. The generator initialises the protocol by assigning to each wire w_i in the circuit two random labels w_i^0 and w_i^1 of length n (the security parameter), representing the respective Boolean values 0 and 1. For each gate the generator computes a *garbled truth table*. Each table consists of four encrypted entries of the output wire labels w_o^γ. These are encrypted according to the gate's Boolean functionality $g(\alpha, \beta)$ using the input wire labels w_l^α and w_r^β as keys. Thus, an entry in the table is encrypted as

$$E_{w_l^\alpha}(E_{w_r^\beta}(w_o^{g(\alpha,\beta)})).$$

After their creation, the garbled tables are randomly permuted and sent to the evaluator, who, so far, is unable to decrypt a single row of any garbled table due to the random choice of the wire labels. To initiate the circuit evaluation, the generator sends its input bits x in form of his input wire labels to the evaluator. Moreover, the wire labels corresponding to the evaluator's input y are transferred via an OT protocol, with the generator being the OT sender, who inputs the two wire labels, and evaluator being the OT receiver, who inputs its input bits of the computation. After the OT step, the evaluator is in possession of the garbled circuit and one input label per input wire. With this information the evaluator is able to iteratively decrypt the circuit from input wires to output wires. Once all gates are evaluated, all output wire labels are known to the evaluator. In the last step of the protocol, the generator sends an output description table to the evaluator, containing a mapping between output label and actual bit value. The decrypted output is then shared with the generator.

We refer to [LP09] for a detailed description as well as a rigorous security proof assuming passive adversaries. To automatically turn an arbitrary function into an STC protocol, one can use the compiler of [Hol+12], which takes the description of a function in ANSI C and produces a Boolean circuit, that can be passed to an STC framework such as [Hua+11] to obliviously compute the desired function.

Chapter 4

A New ORAM Framework

Lax security proofs for ORAMs have been the norm in the past years, although the security model introduced by Goldreich and Ostrovsky constitutes one of the most rigorous and mature security definitions in the field of cryptography. In this chapter, we close the gap between the strict definition of Goldreich and Ostrovsky and the high-level security proofs that have appeared in the literature in the past years, and establish the necessary theoretical background that will allow us to study and argue about the security and privacy guarantees of the constructions that we will develop. We do this by addressing the following question.

Can we bridge the gap between Goldreich and Ostrovsky's formal ORAM definition, and the rather informal approach of ORAM definitions used in the literature, in a way that still maintains a certain degree of mathematical rigour while at the same time being simple and flexible to use?

We answer the above question in a positive way, by providing a framework which is very flexible in describing ORAM constructions, and yet allows us to rigorously define ORAM security and argue about it using game-based techniques. In fact, we will make heavy use of this framework in the following chapters, where we will introduce complex ORAM architectures. Through our new framework we will be able to argue about the security and privacy guarantees in a formal, yet very intuitive way.

Published Content

The results from this chapter first appeared in [GKK17], which is a joint work with Tommaso Gagliardoni and Stefan Katzenbeisser. The development of the framework was done by myself and Tommaso Gagliardoni, and the ORAM security definition has thus also appeared in [Gag17].

4.1. ORAM Definition

On a high level, an Oblivious Random Access Machine (ORAM) is a machine for which the sequences of accessing its memory locations are indistinguishable for any two inputs of equal length and equal running time. The existence of such machines was first examined by Goldreich in his seminal work [Gol87], and the model of computation was later formally defined by Goldreich and Ostrovsky in [GO96], while the first constructions of such architectures were proposed.

Formally defining an ORAM is a demanding task. In fact, Goldreich and Ostrovsky's work, although it provided a rigorous and necessary background for any ORAM architecture, has rarely been used by the community as a means of proving the security of ORAM constructions. Due to the heavily involved nature of Goldreich and Ostrovsky's model, the community (in most of the cases we are aware of) resorted to less complex and more 'intuitive' models which however have not always been mathematically sound. This has led to cases where whole families of ORAM constructions have been shown insecure, as [KLO12]. Unfortunately, the situation did not improve significantly after these revelations, and even recently, Nayak et al. [Nay+16] proved that ORAMs previously though to be secure, like C-ORAM [MMB15] and Chf-ORAM [MBM15], have serious security flaws.

To resolve this situation, we believe that a more versatile ORAM model has to be introduced. Our aim is to find a balance between the heavy formalism of Goldreich and Ostrovsky's model, and the oversimplified models which have been gradually established in the ORAM community. We do this in this chapter, where we present an ORAM model that can describe every ORAM architecture that we are aware of.

We consider two PPT Turing machines that share a common communication tape Ξ: a client $\mathtt{C}$ and a server $\mathtt{S}$. The client is in possession of private data, split in a fixed number of *datablocks*. Each such datablock is an area of the client's memory and is associated with a *unique identifier* id which can be efficiently computed by the client using constant amount of space. This can be done in a variety of ways. However, for ease of exposition and without loss of generality, we will assume in the following that for a set of N datablocks, each datablock's identifier is a number between 1 and N. This way, we will consider that each client's datablock consists of the actual data (which is of size B bits), and the identifier of size $\log N$ bits[1]. Thus, we will consider each client's datablock being of a total size of $D = B + \log N$ bits. By block_i we will denote the datablock consisting of B bits of data, and $\log N$ bits for the identifier i.

The client stores each datablock block_u (usually encrypted under a semantically secure encryption scheme) in a data structure that we will call *the server's database* $\mathtt{DB}$. Note here, that by 'database', we do not impose any structure or any operations on the server's memory, and thus this 'database' should not be confused with other organized storage structures such as relational databases. Our only requirement is that the server stores the datablocks using a logical schema which is imposed by the client. The client

[1] One might wonder, why the bits needed for the identifier should be dependent on the number of datablocks stored. In reality this is not the case, since one could simply ask for a constant number of b bits (e.g., the output of a hash function) for the identifier and avoid this dependency. However, we believe that making this confinement helps intuitively understanding the model, although this trade-off does not signigicantly reduce the mathematical rigour of our description.

stores his datablocks on the server's database, and accesses them for reading or writing by means of *data requests*, which we define in the following.

Definition 8 (Data Request)**.**
A data request to a database $\mathtt{S.DB}$ *of size* N *is a tuple* $\mathtt{dr} = (\mathsf{op}, i, \mathsf{data})$*, where* $\mathsf{op} \in \{\mathtt{Read}, \mathtt{Write}\}, i \in \{1, \ldots, N\}$*, and* $\mathsf{data} \in \{0,1\}^B$ *is a datablock (*data *can also be* $\perp$ *if* $\mathsf{op} = \mathtt{Read}$*).*

ORAM is an interactive protocol between the two machines, and thus each data request results in information being written on the client's and the server's shared communication tape. Since an ORAM is a multi-round interactive protocol, we will consider com as a discrete function of the protocol's round $1, 2, \ldots$, and can thus define the *communication transcript* at a specific round of the protocol which we will further use to define the server's *view*.

Definition 9 (Communication Transcript)**.**
A communication transcript com_t *at round* t *of the protocol is the content of the communication channel* Ξ *at round* t *of the protocol's execution.*

Definition 10 (View of the Server)**.**
Let $\leq$ *denote a partial order over the set of communication transcripts* $A = \{\mathsf{com}_1, \mathsf{com}_2, \ldots, \mathsf{com}_T\}$ *for an ORAM protocol of* T *rounds, i.e., for any* $\mathsf{com}_i \in A$, $\mathsf{com}_j \in A$, $\mathsf{com}_z \in A$ *the following holds:*

1. *For any* $\mathsf{com}_i \in A$, $\mathsf{com}_i \leq \mathsf{com}_i$.
2. *For any* $\mathsf{com}_i \in A$, $\mathsf{com}_j \in A$, *if* $\mathsf{com}_i \leq \mathsf{com}_j$ *and* $\mathsf{com}_j \leq \mathsf{com}_i$, *then* $\mathsf{com}_i = \mathsf{com}_j$.
3. *For any* $\mathsf{com}_i \in A$, $\mathsf{com}_j \in A$, $\mathsf{com}_z \in A$, *it holds that if* $\mathsf{com}_i \leq \mathsf{com}_j$ *and* $\mathsf{com}_j \leq \mathsf{com}_z$, *then* $\mathsf{com}_i \leq \mathsf{com}_z$.

The server's view is the union of all communication transcripts from round 1, to T*:* $\mathtt{View} = \cup_{t=1}^{T} \mathsf{com}_t$.

Depending on the ORAM construction, various cryptographic primitives might be needed to fully describe each architecture (for example the hierarchical ORAM of [GO96] utilises a symmetric encryption scheme and hash functions, while the tree ORAM of [Ste+13a] utilises a symmetric encryption scheme and a pseudorandom number generator). However, every ORAM construction that we are aware of, uses a secret- or public-key encryption scheme under which every datablock of the client is encrypted. We will thus assume this encryption scheme as the minimal requirement in our ORAM definition. We assume that the server's database originally contains only '0', and for N client's datablocks, the server's database is capable of holding N^c encrypted client datablocks, for some constant $c > 1$. During an initialization phase, the client uploads his datablocks to $\mathtt{S}$; how this initial upload is done does not need to be defined here, since there are many ways of how this can be accomplished. The most intuitive way of performing this, is having the client making a series of '$\mathtt{Write}$' operations. Once $\mathtt{S.DB}$ is initialized, the client accesses his datablocks by performing data requests. After each such data request, the client's and the server's states as well as the communication transcript are updated. Having these in mind, we can now define ORAM by means of the following definition.

Definition 11 (ORAM)**.**
Let $\tilde{N} \in \mathbb{N}, M \geq D$ *and* $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ *be a secret-key encryption scheme*[2] *mapping* M*-bit plaintexts to* B*-bit ciphertexts. An ORAM* $\mathtt{ORAM}_{\mathcal{E}}$ *with parameters* $(D, \tilde{N}, \mathcal{E})$ *is a pair of two-party interactive randomised algorithms,* $(\mathtt{ORAM.Init}, \mathtt{ORAM.Access})$*, such that:*

- $\mathtt{ORAM.Init}(\lambda, N) \rightarrow (\mathsf{C}, \mathsf{S})$ *in the following way:*
  1. λ *is the security parameter,* $N < \tilde{N}$*;*
  2. $k \leftarrow \mathtt{KeyGen}(1^{\lambda})$ *is generated by* C*;*
  3. S *includes a database* $\mathsf{S}.DB = (\mathsf{block}_1, \ldots, \mathsf{block}_{\tilde{N}})$*;*
- $\mathtt{ORAM.Access}(\mathsf{C}, \mathsf{S}, \mathtt{dr}) \rightarrow (\mathsf{C}', \mathsf{S}', \mathsf{com})$ *in the following way:*
  1. C *issues a data request* $\mathtt{dr}$*;*
  2. C *and* S *communicate through* Ξ *and produce the communication's transcript* com*;*
  3. *After execution of* $\mathtt{dr}$*, the internal state of* C *is changed to* C'*, and the internal state of* S *is changed to* S'*.*

Until now we have abstractly defined ORAM as a protocol ran between the two machines, C and S, and have not mentioned anything regarding an ORAM's security and soundness. We will define ORAM security in Section 4.2. Regarding the ORAM soundness we will avoid delving into great depths in order to define it, as we focus more on the security aspects of ORAM. We attempt however a high-level overview of ORAM soundness. Practically the main invariant for an ORAM architecture in order to be *sound* (or *complete*), is that the client can always find the datablock he is looking for. Again we stress that different ORAM constructions might have additional requirements or restrictions depending on the scenario and usage – for example in the case of Path-ORAM, the client's stash should never overflow, however the main invariant remains the same in all constructions.

In any ORAM construction, after every access, the client changes the server's database internal state, in a way that the server cannot extract any information about the state itself. This is usually done by some encrypted information which is available only to the client. Thus, a minimal soundness condition is that particular information is 'accessible' by the client. Since we want to include in this 'accessibility' any way that a particular piece of information is made available to the client, we will say that an element x is *accessible* by a machine M (e.g., the client) if M can simulate the constant oracle $O_x(.) \mapsto x$. With these in mind, we define the *minimal ORAM soundness conditions* as follows.

Definition 12 (Minimal ORAM Soundness Conditions)**.**
An ORAM construction $\mathtt{ORAM}_{\mathcal{E}}$ *has minimal soundness if the following hold:*

1. *for any* (λ, N)*, if* $(\mathsf{C}, \mathsf{S}) \leftarrow \mathtt{ORAM.Init}(\lambda, N)$*, then the secret key* k *from Def. 11 must be accessible by* C*;*

[2] For ease of exposition, we analyse here only the case of ORAM equipped with a secret-key encryption schemes. However, the same ideas can be applied in defining an ORAM using public-key encryption scheme.

2. *for any* $\mathtt{dr} = (\mathsf{op}, i, \mathsf{data})$*, if* $(C', S', \mathsf{com}) \leftarrow \mathtt{ORAM.Access}(\mathsf{C}, \mathsf{S}, \mathtt{dr})$*, then:*

   a) *if the secret key* k *is accessible by* C*, then* k *is also accessible by* C'*;*

   b) *if* $\mathsf{op} = \textit{read}$ *and* $S.DB(i) = \mathsf{block}$*, then* $\mathsf{block.Data}$ *must be accessible by* C'*;*

   c) *if* $\mathsf{op} = \textit{write}$ *and* $S'.DB(i) = \mathsf{block}$*, then* $\mathsf{block.Data} = \mathsf{data}$.

Note that in the above definition we do not state anything regarding the accessibility of the server to the key k or to any other information. After all, a soundness definition takes into account only honest behaving parties, and in such case, since the server is there only to access logical addresses of his memory as indicated by the client, the soundness is maintained. What happens if the server starts to deviate from the protocol, is a property examined by the *security* of the protocol, and which we will investigate in the next section.

4.2. ORAM Security

Historically seen, ORAM architectures are usually complex and as such it is intuitive to first define their security against the more relaxed *semi-honest* adversaries [Gol04]. In this threat model, the adversary $\mathtt{A}$ is a PPT machine that has complete control over the communication channel and over the inputs of the protocol that is under attack. However, $\mathtt{A}$ does not alter the contents of the communication channel, since this would jeopardise the protocol's soundness. In the case of ORAMs, $\mathtt{A}$ tries to deduce information about the client's accesses, by monitoring the protocol's transcript and by possibly making accesses himself. Thus, without loss of generality one can assume that in a single client setting, the adversary *is* the server. The information that can be leaked to the adversary might, for example, include how often the client accesses a specific datablock, if particular datablocks are more important than others (because they have been accessed more often), or even a deduction of a behavioural pattern, e.g., after a certain number of accesses to specific datablocks, the client logs out of the system. We can thus define the adversary that we will take into account in the rest of this work, as follows.

Definition 13 (ORAM Adversary)**.**
An ORAM adversary $\mathtt{A}$ *is a PPT algorithm which has complete control of* S*, as long as the ORAM's soundness is preserved.*

In order to abstractly describe what such ORAM adversaries can do, the notion of *access patterns* was introduced. On a high level, an access pattern consists practically of all the changes on the server's database, and all the information exchanged between the client and the server during the course of each protocol's round (in the literature sometimes also referred to as *adversarial views*) after a client's call to $\mathtt{ORAM.Access}$. We now define the notion of access pattern more formally.

Definition 14 (Access Pattern)**.**
Given an ORAM client C*, a server* S*, and a data request* $\mathtt{dr}$*, the access pattern* $\mathtt{ap}(\mathtt{dr})$ *is the tuple* $\left(S.DB, \mathsf{com}, S'.DB\right)$*, where* $(C', S', \mathsf{com}) \leftarrow \mathtt{ORAM.Access}(\mathsf{C}, \mathsf{S}, \mathtt{dr})$.

An ORAM is considered secure if during the client's accesses no information about the client's inputs is leaked to the server. One way of arguing about the inexistence of such leakage is by showing that the access patterns induced by the same number of data requests are computationally indistinguishable for anyone but the client. This formulation reminds of the definition of *semantic security* for encryption schemes [Gol04] and as such, we will consider here an adaptive game-based indistinguishability notion played between the adversary and the client. The adversary sends two data requests to the client; the client chooses one of them and executes it. The adversary (having complete control of the network, as well as the server's database) sees the induced access pattern and tries to find out which of the two data requests was executed, and wins if he found the correct one. This is formally described by the following indistinguishability game.

Definition 15 ($\mathsf{Game}^{\mathtt{AP\text{-}IND\text{-}CQA}}_{\mathtt{A},\mathtt{ORAM}}(\lambda)$)**.**
Let $\mathtt{ORAM} = (\mathtt{ORAM.Init}, \mathtt{ORAM.Access})$ *be an ORAM construction,* λ *a security parameter and* $\mathtt{A}$ *an ORAM adverary. The computational indistinguishability of access patterns game under adaptive chosen query attack* $\mathsf{Game}^{\mathtt{AP\text{-}IND\text{-}CQA}}_{\mathtt{A},\mathtt{ORAM}}(\lambda)$ *proceeds as follows:*

1. $\mathtt{A}$ *chooses* $N \leq \tilde{N}$*;*
2. $(C, S) \leftarrow \mathtt{ORAM.Init}(\lambda, N)$*;*
3. *First CQA learning phase: for* $i = 1, \ldots, q_1 \in \mathbb{N}$*,* $\mathtt{A}$ *repeats (adaptively) the following:*
   a) $\mathtt{A}$ *chooses a data request* $\mathtt{dr}_i$*;*
   b) C *executes* $\mathtt{ORAM.Access}$ *on* $\mathtt{dr}_i$*;*
   c) $\mathtt{A}$ *receives* $\mathtt{ap}(\mathtt{dr}_i)$*;*
4. *Challenge phase:* $\mathtt{A}$ *chooses two data requests* $\mathtt{dr}^0$ *and* $\mathtt{dr}^1$*;*
5. C *flips a random secret bit* $b \xleftarrow{\$} \{0, 1\}$ *and executes* $\mathtt{ORAM.Access}$ *on* $\mathtt{dr}^b$*;*
6. $\mathtt{A}$ *receives* $\mathtt{ap}(\mathtt{dr}^b)$*;*
7. *Second CQA learning phase: for* $j = 1, \ldots, q_2 \in \mathbb{N}$*,* $\mathtt{A}$ *repeats (adaptively) the following:*
   a) $\mathtt{A}$ *chooses a data request* $\mathtt{dr}_j$*;*
   b) C *executes* $\mathtt{ORAM.Access}$ *on* $\mathtt{dr}_j$*;*
   c) $\mathtt{A}$ *receives* $\mathtt{ap}(\mathtt{dr}_j)$*;*
8. $\mathtt{A}$ *outputs a bit* b'.

$\mathtt{A}$ *wins the game iff* $b = b'$.

Definition 16 (Access Pattern Indistinguishability Under Adaptive Chosen Query Attack)**.**
An ORAM construction ORAM *has computationally indistinguishable access patterns under adaptive chosen query attack (or, it is AP-IND-CQA-secure) if and only if for any ORAM adversary* A*:*

$$\left|\text{Prob}\left[\,\mathsf{Game}^{AP\text{-}IND\text{-}CQA}_{A,\mathtt{ORAM}}(\lambda)=1\right]-\frac{1}{2}\right|\leq negl(\lambda).$$

4.2.1. Equivalence to the Simulation Model and Advantages

Traditionally in the ORAM literature, security has been considered in terms of indistinguishability of access patterns but in a non-adaptive fashion. That is, the adversary chooses two different data requests tuples of the same length, and has to distinguish between the access patterns produced by the client executing one of the two, chosen at random. Our model has the advantage that it also considers adversaries who play this game changing adaptively their sequences of data requests.

In a concurrent and independent work Garg et al. [GMP16], introduced another, simulation-based security definition. Their definition states that for any ORAM adversary, it must be computationally hard to distinguish between the access pattern distributions produced by a real client and by a simulator producing bogus transcripts, even if the adversary is allowed to choose adaptively the data requests to be executed by the real client. Although this definition provides strong security guarantees, still it is not simple enough in order to be used in complex ORAM constructions like the ones we will introduce in the following. Yet, we show that our novel definition is equivalent to the simulation-based notion given in [GMP16], according to which no computationally bounded adversary can distinguish between the interaction with a real client and the interaction with a simulator that produces bogus transcripts. Our security notion AP-IND-CQA is therefore at least as strong as all other security notions for ORAM introduced to date, but it has the advantage of being game-based (and hence easier to deal with in security reductions).

Definition 17 (Access Pattern Simulability Under Adaptive Chosen Query Attack)**.**
An ORAM construction ORAM *has simulable access patterns under adaptive chosen query attack (or, it is AP-SIM-CQA-secure) iff for any classical ORAM adversary* A *the following two distributions are computationally indistinguishable:*

1. $\mathtt{ap}(\mathtt{dr} \leftarrow \mathtt{A})$;
2. $\mathtt{ap}(\mathtt{dr} \xleftarrow{\$} \{\text{‘read’,‘write’}\} \times \{1,\ldots,N\} \times \{0,1\}^D)$.

Proposition 1.
An ORAM construction ORAM *is AP-SIM-CQA secure iff it is AP-IND-CQA secure.*

The idea of the proof is to go through a hybrid argument in the same way that shows IND-CPA security for encryption schemes to be equivalent to Real-or-Random security (see for example [Bel+97]).

Proof. The proof closely follows the reduction for equivalent notions of IND-CPA security found in [Bel+97]. As a first step, we rephrase Definition 17 in terms of a security game – the two resulting definitions of AP-SIM-CQA security are obviously equivalent.

Definition 18 ($\mathsf{Game}^{\texttt{AP-SIM-CQA}}_{\texttt{A,ORAM}}(n)$)**.**
Let $\texttt{ORAM} = (\texttt{ORAM.Init}, \texttt{ORAM.Access})$ *be an ORAM construction,* n *a security parameter and* $\texttt{A}$ *an ORAM adverary. The computational simulability of access patterns game under adaptive chosen query attack* $\mathsf{Game}^{\texttt{AP-SIM-CQA}}_{\texttt{A,ORAM}}(n)$ *proceeds as follows:*

1. $\texttt{A}$ *chooses* $N \leq \tilde{N}$*;*
2. $(\texttt{C}, \S) \leftarrow \texttt{ORAM.Init}(n, N)$*;*
3. $\texttt{C}$ *flips a secret random bit* $b \xleftarrow{\$} \{0,1\}$*;*
4. $\texttt{A}$ *repeats (adaptively) the following, for* $i = 1, \ldots, q$*:*
   a) $\texttt{A}$ *chooses a data request* $\texttt{dr}_i$*;*
   b) *if* $b = 0$*, then* $\texttt{C}$ *sets* $\texttt{dr} := \texttt{dr}_i$*;*
   c) *else if* $b = 1$*, then* $\texttt{C}$ *sets* $\texttt{dr} \xleftarrow{\$} \{$ ‘read’,‘write’ $\} \times \{1, \ldots, N\} \times \{0,1\}^D$*;*
   d) $\texttt{C}$ *executes* $\texttt{ORAM.Access}$ *on* $\texttt{dr}$*;*
   e) $\texttt{A}$ *receives* $\texttt{ap}(\texttt{dr}_i)$*;*

Finally, $\texttt{A}$ *outputs a bit* b'*, and wins the game iff* $b = b'$*.*

Definition 19 ((Game-Based) Access Pattern Simulability Under Adaptive Chosen Query Attack)**.**
An ORAM construction $\texttt{ORAM}$ *has simulable access patterns under adaptive chosen query attack (or, it is AP-SIM-CQA-secure) iff for any classical ORAM adversary* $\texttt{A}$*:*

$$\left| \text{Prob}\left[\mathsf{Game}^{\texttt{AP-SIM-CQA}}_{\texttt{A,ORAM}}(n) = 1 \right] - \frac{1}{2} \right| \leq \textit{negl}(n).$$

Then, we prove Proposition 1 by double implication.

Lemma 4.2.1.1 (AP-SIM-CQA $\implies$ AP-IND-CQA)**.**

Proof of Lemma 4.2.1.1. Let $\mathcal{A}^{\texttt{IND}}$ be an ORAM adversary such that

$$\text{Prob}\left[\mathsf{Game}^{\texttt{AP-IND-CQA}}_{\mathcal{A}^{\texttt{IND}},\texttt{ORAM}}(n) = 1 \right] = \frac{1}{2} + \nu(n),$$

for a non-negligible function ν. We will use $\mathcal{A}^{\texttt{IND}}$ in a black-box way to construct another ORAM adversary $\mathcal{A}^{\texttt{SIM}}$, able to break the AP-SIM-CQA security of $\texttt{ORAM}$, against the assumption.

At the beginning, $\mathcal{A}^{\mathtt{SIM}}$ runs $\mathcal{A}^{\mathtt{IND}}$, which starts $\mathsf{Game}^{\mathtt{AP\text{-}IND\text{-}CQA}}_{\mathcal{A}^{\mathtt{IND}},\mathtt{ORAM}}$ by choosing n and N. At the same time, $\mathcal{A}^{\mathtt{SIM}}$ starts $\mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$ with the same parameters.

During the first IND learning phase, whenever $\mathcal{A}^{\mathtt{IND}}$ performs a CQA query $\mathtt{dr}_i$, $\mathcal{A}^{\mathtt{SIM}}$ responds to such a query by forwarding it to its SIM (Real-or-Random) challenger, and then forwarding to $\mathcal{A}^{\mathtt{IND}}$ the related access pattern. When $\mathcal{A}^{\mathtt{IND}}$ performs its challenge query of the form $(\mathtt{dr}^0, \mathtt{dr}^1)$, $\mathcal{A}^{\mathtt{SIM}}$ flips a random bit $b^* \xleftarrow{\$} \{0,1\}$, forwards $\mathtt{ap}^{b^*}$ to the SIM challenger, and then forwards to $\mathcal{A}^{\mathtt{IND}}$ the related access pattern. Then, during the second IND learning phase, $\mathcal{A}^{\mathtt{SIM}}$ behaves like in the first challenge phase, by forwarding the queries to its SIM challenger and returning the access patterns to $\mathcal{A}^{\mathtt{IND}}$. Finally, when $\mathcal{A}^{\mathtt{IND}}$ outputs a bit b', if $b' = b^*$ then $\mathcal{A}^{\mathtt{SIM}}$ outputs 0, otherwise $\mathcal{A}^{\mathtt{SIM}}$ outputs a random bit.

The reduction works for the following reason: let us assume that $b = 0$, i.e., the SIM challenger was a honest ORAM client $\mathtt{C}$. Then, during the whole reduction, $\mathcal{A}^{\mathtt{SIM}}$ successfully simulated a real client for $\mathcal{A}^{\mathtt{IND}}$, in an AP-IND-CQA game where the secret bit was b^*. By assumption, $\mathcal{A}^{\mathtt{IND}}$ guesses correctly such bit with probability at least $\frac{1}{2} + \nu(n)$. In that case, also $\mathcal{A}^{\mathtt{SIM}}$ wins, so:

$$\mathrm{Prob}\left[\mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}(n) = 1 \,\middle|\, b = 0\right] \geq \frac{1}{2} + \nu. \tag{4.1}$$

On the other hand, if $b = 1$ (i.e., the SIM challenger was simulating fake access patterns) we cannot say anything on $\mathcal{A}^{\mathtt{IND}}$'s success probability, because for the whole time it has played in a malformed game. But we can say that, even if $\mathcal{A}^{\mathtt{IND}}$ fails, $\mathcal{A}^{\mathtt{SIM}}$ still succeeds with probability $\geq \frac{1}{2} - \varepsilon$ for a negligible ε:

$$\mathrm{Prob}\left[\mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}(n) = 1 \,\middle|\, b = 1\right] \geq \frac{1}{2} - \varepsilon. \tag{4.2}$$

Thus, combining 4.1 and 4.2, the reduction's overall success probability is:

$$\mathrm{Prob}\left[\mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}(n) = 1\right] \geq \frac{1}{2} + \frac{\nu}{2} - \varepsilon,$$

which concludes the proof. □

Lemma 4.2.1.2 (AP-IND-CQA $\implies$ AP-SIM-CQA)**.**

Proof of Lemma 4.2.1.2. Let $\mathcal{A}^{\mathtt{SIM}}$ be an ORAM adversary such that

$$\mathrm{Prob}\left[\mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}(n) = 1\right] \geq \frac{1}{2} + \nu(n),$$

for a non-negligible function ν. We will use $\mathcal{A}^{\mathtt{SIM}}$ in a black-box way to construct another ORAM adversary $\mathcal{A}^{\mathtt{IND}}$, able to break the AP-IND-CQA security of $\mathtt{ORAM}$, against the assumption. The proof uses a hybrid argument on the number q of queries performed by $\mathcal{A}^{\mathtt{SIM}}$ during $\mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$.

More in detail: $\mathcal{A}^{\mathtt{IND}}$ runs $\mathcal{A}^{\mathtt{SIM}}$, which starts $\mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$ by choosing n and N. At the same time, $\mathcal{A}^{\mathtt{IND}}$ starts $\mathsf{Game}^{\mathtt{AP\text{-}IND\text{-}CQA}}_{\mathcal{A}^{\mathtt{IND}},\mathtt{ORAM}}$ with the same parameters, and also chooses an index $j \xleftarrow{\$} \{1,\ldots,q\}$ uniformly at random. Whenever $\mathcal{A}^{\mathtt{SIM}}$ performs a query with a data request $\mathtt{dr}_i$, $\mathcal{A}^{\mathtt{IND}}$ does the following:

- For the first $j-1$ queries, $\mathcal{A}^{\mathtt{IND}}$ executes $\mathtt{dr}_i$ using his own oracle in the first CQA learning phase, and responds with $\mathtt{ap}(\mathtt{dr}_i)$.
- At the j-th query, $\mathcal{A}^{\mathtt{IND}}$ does the following:
  - samples at random $\mathtt{dr}^* \xleftarrow{\$} \{\text{'read'},\text{'write'}\} \times \{1,\ldots,N\} \times \{0,1\}^D$;
  - sets $\mathtt{dr}^0 := \mathtt{dr}_j$ and $\mathtt{dr}^1 := \mathtt{dr}^*$.
  - performs his AP-IND-CQA challenge query $(\mathtt{dr}_0, \mathtt{dr}_1)$, and responds to $\mathcal{A}^{\mathtt{SIM}}$ with $\mathtt{ap}(\mathtt{dr}^b)$.
- Starting from the $(j+1)$-th query, $\mathcal{A}^{\mathtt{IND}}$ ignores $\mathcal{A}^{\mathtt{SIM}}$'s data requests, and always responds with access patterns produced by freshly generated random data requests using his own oracle in the second CQA learning phase.

Finally, when $\mathcal{A}^{\mathtt{SIM}}$ outputs a bit b', if $b' = 0$ then $\mathcal{A}^{\mathtt{IND}}$ outputs 0, otherwise $\mathcal{A}^{\mathtt{IND}}$ outputs a random bit. We show that:

$$\mathrm{Prob}\left[\mathsf{Game}^{\mathtt{AP\text{-}IND\text{-}CQA}}_{\mathcal{A}^{\mathtt{IND}},\mathtt{ORAM}}(n) = 1\right] \geq \frac{1}{2} + \frac{\nu}{2q}.$$

The hybrid argument goes as follows. For $i = 0,\ldots,q$ consider a sequence of intermediate hybrid games $\mathsf{Game}^{\mathtt{HYB\text{-}i}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$, where $\mathsf{Game}^{\mathtt{HYB\text{-}i}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$ is defined as the usual $\mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$ by executing through the SIM (Real-or-Random) oracle only the first i data requests output by $\mathcal{A}^{\mathtt{SIM}}$, and executing random data requests afterwards. Notice that:

- $\mathsf{Game}^{\mathtt{HYB\text{-}q}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$ coincides with $\mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$, and
- $\mathsf{Game}^{\mathtt{HYB\text{-}0}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$ coincides with $\mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$ in the case that $b = 1$.

Moreover, the following hold:

- the output b' of $\mathcal{A}^{\mathtt{SIM}}$ in the above reduction coincides with the output of $\mathcal{A}^{\mathtt{SIM}}$ in $\mathsf{Game}^{\mathtt{HYB\text{-}(i+1)}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$ if $b = 0$, while
- the output b' of $\mathcal{A}^{\mathtt{SIM}}$ in the above reduction coincides with the output of $\mathcal{A}^{\mathtt{SIM}}$ in $\mathsf{Game}^{\mathtt{HYB\text{-}i}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}$ if $b = 1$.

At this point, as j was chosen uniformly at random in $\{1,\ldots,q\}$, we can write the advantage of $\mathcal{A}^{\mathtt{IND}}$ in $\mathsf{Game}^{\mathtt{AP\text{-}IND\text{-}CQA}}_{\mathcal{A}^{\mathtt{IND}},\mathtt{ORAM}}(n)$ by use of the triangular inequality as follows:

$$\mathrm{Prob}\Big[\mathcal{A}^{\mathtt{IND}} \text{ outputs } 1 \Big| b=1\Big] - \mathrm{Prob}\Big[\mathcal{A}^{\mathtt{IND}} \text{ outputs } 1 \Big| b=0\Big] =$$

$$\frac{1}{q}\sum_{i=0}^{q-1}\left(\mathrm{Prob}\left[\mathcal{A}^{\mathtt{SIM}} \text{ outputs } 1 \text{ in } \mathsf{Game}^{\mathtt{HYB\text{-}i}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}\right] - \mathrm{Prob}\left[\mathcal{A}^{\mathtt{SIM}} \text{ outputs } 1 \text{ in } \mathsf{Game}^{\mathtt{HYB\text{-}(i+1)}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}\right]\right) =$$

$$\frac{1}{q}\left(\mathrm{Prob}\left[\mathcal{A}^{\mathtt{SIM}} \text{ outputs } 1 \text{ in } \mathsf{Game}^{\mathtt{HYB\text{-}0}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}\right] - \mathrm{Prob}\left[\mathcal{A}^{\mathtt{SIM}} \text{ outputs } 1 \text{ in } \mathsf{Game}^{\mathtt{HYB\text{-}q}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}\right]\right) =$$

$$\frac{1}{q}\left(\mathrm{Prob}\left[\mathcal{A}^{\mathtt{SIM}} \text{ outputs } 1 \text{ in } \mathsf{Game}^{\mathtt{HYB\text{-}0}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}} \Big| b=1\right] - \mathrm{Prob}\left[\mathcal{A}^{\mathtt{SIM}} \text{ outputs } 1 \text{ in } \mathsf{Game}^{\mathtt{HYB\text{-}q}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}\right]\right) \geq$$

$$\frac{1}{2q}\left[\text{ advantage of } \mathcal{A}^{\mathtt{SIM}} \text{ in } \mathsf{Game}^{\mathtt{AP\text{-}SIM\text{-}CQA}}_{\mathcal{A}^{\mathtt{SIM}},\mathtt{ORAM}}\right] = \frac{\nu}{2q}$$

which concludes the proof. □

The combination of Lemma 4.2.1.1 and Lemma 4.2.1.2 concludes the proof of Proposition 1.

□

4.3. Application to Path-ORAM

An important building block for one of our constructions is Path-ORAM [Ste+13a] by Stefanov's et al. In [Ste+13b] a detailed proof of Path-ORAM's soundness is given. However, a full formal proof of Path-ORAM's security is not provided, and is rather presented on a relatively high and informal level. Using our new formalism, we have been able to provide Path-ORAM's *first* formal security proof. Here, we describe on a high level how Path-ORAM works, and refer to [Ste+13a] for more implementation details. We then describe Path-ORAM using our new formalism, and give an overview of its full security proof, while for details of the latter we refer to [Gag17]

Using our new formalism, we give here a full description of PathORAM (which from now on we denote as `Path-ORAM`). We set λ to be the security parameter, used by the encryption scheme $\mathcal{E}$, and N the maximum number of 'real' datablocks that `C` wants to store. As we have seen already, in `Path-ORAM`, `S` stores a total of $\tilde{N}$ datablocks, which are linear in N and thus $N \leq \tilde{N}$. Observe here, that unlike $\tilde{N}$, N can be dictated by the adversary in the security game. With Z, we will denote the maximum number of datablocks that can be stored in every tree node. This parameter is architecture dependent and guarantees

the protocol's soundness: the smaller Z is, the higher the probability that C will have to use his stash. As it has been (also experimentally) shown in [Ste+13a], setting $Z = 5$ is usually sufficient for the stash to remain small. However, we treat here Path-ORAM abstractly, and thus we do not need to fix Z, since any nonzero value should work in theory. By T we denote the minimum number of bits that are needed to index all the 'real' N datablocks. Observe at this point that without loss of generality, we can assume that indexing all $\tilde{N}$ datablocks stored on S is not needed, as every 'fake' datablock can be indexed by a fixed bitstring (like for example 0^k). D is the bitlength of the datablocks data used in the Path-ORAM implementation, and it is thus architecture-dependant. M is the total bitlength of a datablock (i.e., the data plus the index) and is thus also architecture-dependant. The encryption scheme $\mathcal{E}$ must be able to work with M-bit plaintexts. B is the size of a ciphertext produced by the encryption scheme $\mathcal{E}$, and hence the total size of an encrypted datablock. Thus, the size of S's tree storage memory is at most $B\tilde{N}$ bits.

Algorithm 1: Path-ORAM.Init(λ, N)

1 C generates a secret key $k \leftarrow \mathtt{KeyGen}(1^\lambda)$;
2 set $T = \lceil log_2 N \rceil$; (notice $T \leq K$);
3 C initialises a lookup table (the *position map*) of the form $\big((1, r_1), \ldots, (N, r_N)\big)$, where r_i are T-bit values generated by truncating the first T bits of $\mathcal{G}$'s output;
4 The server's database DB is stored in a binary tree of height T, with leaves $\mathsf{Leaf}_0, \ldots, \mathsf{Leaf}_{2^T-1}$, such that:
- each node of the tree stores up to Z datablocks;
- every datablock of every node is initialized to $\mathtt{Enc}_k(0^K \| 0^D)$.

Definition 20 (Path-ORAM)**.**
For fixed parameters $D, \tilde{N} \in \mathbb{N}$, let $K = \lceil log_2 \tilde{N} \rceil$, $Z \in \mathbb{N}$, $M = D + K$, $B \geq M$. Let $\mathcal{G}$ be a PRNG as from Definition 6 outputting K-bit values, and $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ be a SKE with M-bit plaintexts and B-bit ciphertexts. We define an ORAM construction called Path-ORAM $=$ Path-ORAM$_{\mathcal{E},\mathcal{G}}$ as a set of two interactive protocols Path-ORAM.Init(λ, N), and Path-ORAM.Access$(\mathsf{op}, i, \mathsf{data})$ as described in Algorithms 1 and 2 respectively, between two PPT machines C and S.

Using our new security framework, we are able to prove the security of Path-ORAM. For a detailed proof we refer to [Gag17].

Proposition 2.
Let $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ be an IND-CPA SKE according to Definition 4, and let $\mathcal{G}$ be a PRNG as from Definition 6. Then, Path-ORAM instantiated using $\mathcal{E}$ and $\mathcal{G}$ is an AP-IND-CPA secure ORAM.

Proof. (sketch) By assuming the existence of a PPT adversary A that wins the IND-CQA game of Definition 15 with non-negligible advantage, a PPT algorithm D is constructed that breaks the encryption scheme's semantic security. For the full details of the proof we refer to [Gag17], while here we simply give an overview of how D works. In our reduction, D simulates a Path-ORAM client C who plays against A who in turn simulates the semi-honest Path-ORAM server. Crucial in our reduction is the fact that D

Algorithm 2: `Path-ORAM.Access`$(\mathsf{op}, i, \mathsf{data})$

```
1  C reads from his position map r_i and sends it to S;
2  S sends to C the path DPath from the root of the tree to Leaf_{r_i};
3  remap (i, r_i) to (i, r_i') in the position map of C, where r_i' is a fresh pseudorandom T-bit value
     (generated by truncating the first K − T bits of G's output), obtaining C';
4  for every datablock block in DPath, C' do /* using k, C' decrypts the datablock
     obtain j ∈ {0,1}^K, and the data data_j ∈ {0,1}^D;                          */
5      Dec_k(block) → (j||data_j) ∈ {0,1}^M;
6      if j == i then
7          if op == 'read' then
8              C' reads data_j;
9          end
10         else if op == 'write' then
11             C' sets data_j = data;
               /* block is updated, so that Data(block) = data                 */
12         end
13     end
14     b_swap = 0;
15     C' re-encrypts block // re-randomisation;
16     Node = FindCommonAncestor (DPath, Leaf_{r_i}, Leaf_{r_j});
       /* find in DPath the common parent node Node between Leaf_{r_i} and Leaf_{r_j},
          closer to the leaf level                                              */
17     for every datablock block' in Node do
18         Dec_k(block') → (j'||data');
19         if j' == 'FAKE' then
20             COPY(block, temp_block);
21             MOVE(block', block);
22             MOVE(temp_block, block);
23             b_swap = 1;
24         end
25         Re-encrypt datablock;
26     end
27     if b_swap = 0 then
28         if Node ≠ ROOT then
29             Set Node in one level higher (i.e., the Node's parent);
30             GoTo Step 17;
31         end
32         else
33             Store block in the Stash;
34         end
35     end
36 end
37 C' sends UPath to S;
38 S updates S.DB with UPath;
39 S' ← S;
40 return com = (r_i, DPath, UPath) ;
```

keeps a plaintext version of the server's database S.DB throughout the game, something that is allowed, as D is a simulator of C and thus does not need to adhere to the same restrictions (e.g., regarding his private storage) as the typical ORAM client C.

The simulation starts by A choosing the number of real datablocks N, the size of the node, Z and the security parameter λ. D executes ORAM.Init, *without creating the encryption scheme's secret key*, initialises his position map, and uses this position map in order to populate the respective Path-ORAM binary tree with N leafs. This tree however is kept locally in plain, 'mirroring' the server's database. D starts playing the IND-CPA game against the encryption scheme for the same security parameter λ chosen by A. For this, D obtains oracle access to $\mathcal{E}_k$, for the unknown secret key k. D now using his encryption oracle, encrypts all the datablocks of his plaintext tree *without changing their position within the tree*, thus obtaining the encrypted version of the Path-ORAM tree. At this point, D can fully simulate the Path-ORAM client: Every time D receives a data request from A, D first adds the respective leaf of path, as well all the encrypted datablocks found in this path from his encrypted tree to com. D then updates his position map and moves all the datablocks in the path of the *plaintext tree* accordingly. Using the encryption oracle, D encrypts every datablock of the updated plaintext path, and adds the newly encrypted path to com, which he sends to A.

In the challenge phase, A sends the tuple $(\mathtt{dr}^0, \mathtt{dr}^1)$ to D. A sets $m_a = (i_a | \mathtt{dat}_a)$ for $a \in \{0, 1\}$, with $\mathtt{dat}_a$ being retrieved by looking for the identifier id_a in the plaintext tree. This way, A executes the IND challenge using (m_0, m_1), and receiving back $c = \mathcal{E}_k(m_b)$ for the secret bit b. D now chooses a random bit b^*, and executes $\mathtt{dr}_{b^*}$, however implanting c as an updated datablock during the execution of $\mathtt{dr}_{b^*}$. After this, D continues simulating C in the second CQA phase as before, waiting for A to output his bit $\tilde{b}$. If $\tilde{b} = b^*$, then D outputs b^*, otherwise he outputs a random bit.

Calculating the total probability that D wins in the IND-CPA game, using A by means of the above discussion, we get that D's advantage is non-negligible in λ which is a contradiction, thus concluding the proof. □

Chapter 5

Proxy-ORAM

Laying the theoretical foundation of an ORAM framework in the previous Chapter, allows us now to construct an ORAM that is crafted in a way that a client can securely store his encrypted data on a remote server, and allow third parties to privately access (i.e., as far as the server is concerned) parts of the data. Further, parties retrieving the data should see in all the steps of the protocol only encrypted data. Thus, in this chapter we will deal with the following question.

Can we build an ORAM that allows a third party to partially access a client's outsourced data?

We give a positive answer to the above question, by presenting the first ORAM construction, that under the assumption that all involved parties are semi-honest and non colluding, achieves at the same time several goals. First and foremost, the client who has outsourced his private data, gives only *temporary* and *partial* access to his data to a third party. By 'temporary' we mean that the data is accessible by the third party only for one round of the protocol. At the same time, the client does not need to be constantly online in order to perform any computation or communication intensive operation (such as a reshuffle). In fact, the only time the client needs to be online, is in order to give a temporary access token to the third party. The third party sees *only* the requested data and *only* in encrypted form. Finally, during the accesses, none of the involved parties learns anything more than the number of accesses performed.

Published Content

The results from this chapter first appeared in [Kar+14], which is a joint work with Andreas Peter, Stefan Katzenbeisser, Erik Tews and Kay Hamacher. The development of the architecture was jointly done between myself, Andreas Peter and Stefan Katzenbeisser, and the security proofs were done by myself.

Our ORAM architecture is based on the hierarchical ORAM solution (cf. Chapter 2.1.1) and consists of three major entities: the *client*, the aforementioned third party – to which we will from now on refer as the *investigator*, and the storage service. The storage service is considered as a specialised service which consists of the actual data store, called the *cloud*, and a separate *non-colluding* entity, in the following referred to as the *proxy*, that assists in computational tasks throughout the ORAM's initialisation and

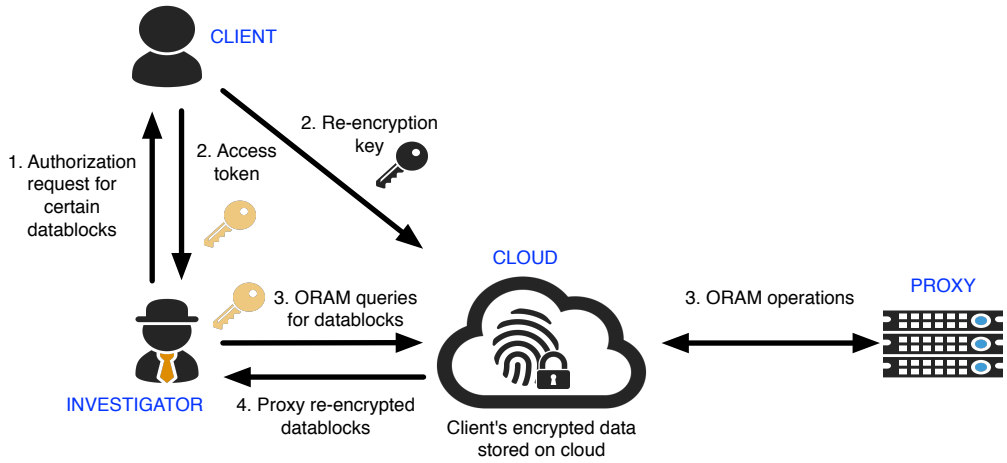


Figure 5.1: Overview of the Proxy-ORAM architecture: The client issues a re-encryption key to the proxy, and a secret key as the access token to the investigator (steps 1 and 2). The investigator issues an ORAM request to the cloud, and the ORAM operations are performed between the cloud and the proxy (step 3). The cloud sends proxy re-encrypted datablocks to the investigator (step 4).

access. Because of this heavy usage of the proxy server, we named our architecture Proxy-ORAM (for a graphical overview of the architecture, cf. to Figure 5.1). Splitting databases into two non-colluding servers is a wide-spread approach in current research, proved to be very promising as for instance shown by Boneh et al. [Bon+13] and Applebaum et al. [App+10].

We achieve giving temporary and partial access to the client's data by using proxy re-encryption as an access control mechanism: The client issues and sends to the investigator a secret key as an access token. The re-encryption key for the investigator's token is sent to the proxy by the client. This way, the encryption of the stored datablocks can be transformed to an encryption that is accessible by the investigator. By temporary access, we mean here that the investigator can only access a datablock once from the server. This way (and as long as all parties are semi-honest and non colluding) nothing can be done without the client's former approval.

By employing the two (non-colluding) servers, and using homomorphic encryption, we achieve complete outsourcing of all computation and communication intensive steps (such as the 'reshuffing') that are present in most ORAM constructions. This way, our architecture allows delegation of access rights, being a completely autonomous storage. In contrast to [Fra+12], it does not require the client to be constantly online.

5.1. Building Blocks

Complying with our ORAM framework description from Chapter 4.1, we assume that the client's data is stored encrypted in small datablocks of fixed size, B bits. Typically for an ORAM construction, fake

datablocks must also reside in the ORAM. Assuming that the client stores N real datablocks, in this construction we add another N fake datablocks. To each fake datablock i we assign a unique, random identifier $\mathsf{id}^i_{\text{fake}}$, making sure that there are no collisions between the fake and real identifiers, and assume further that each identifier has size b bits. Each fake datablock contains random data; when encrypted it must be indistinguishable from an encrypted real datablock. Fake and real datablocks are stored encrypted on the cloud in an $\lceil \log_2 N \rceil$ level pyramid-like structure. In our architecture, the encrypted datablocks are initially uploaded to the last pyramidal level (i.e., a level that can hold all initially uploaded data). After a datablock has been accessed, it is always placed on the first pyramid level. If the first level does not have enough space, then the level is recursively emptied to lower ones, until enough space has been created on the first level – a process that is traditionally called a 'reshuffle'.

In order for the client to find the datablock he wants, he must first find the level on which the datablock currently resides and then the datablock's exact position within that level. Finding the right level is done by utilising Bloom filters: Each pyramidal level is associated to a Bloom filter, whose contents (i.e., the individual filter bits) are kept encrypted on the proxy and which is built interactively between the cloud and the proxy, whenever a level is reshuffled. Within each level, each datablock's virtual address on the server's database is formed by means of a unique, yet mutable 'index', which is identifier-dependent. We shall refer to this datablock's 'index' in the following, as `index`. Thus, once the level where the datablock currently resides has been identified, the investigator retrieves it from that level, using the datablock's `index`.

Since we do not want to involve the client at all during the creation of the encrypted Bloom filter, the latter has to be created interactively between the cloud and the proxy in a way that does not leak any information about the datablock identifiers stored. One way of doing this is by utilising a hash function that operates on shares, such as the Chaum-van Heijst-Pfitzmann hash function (cf. Chapter 3.2.1). One share lies by the proxy, and the other share is stored encrypted alongside with the encrypted datablock. The datablock's `index` follows a similar design: It has to be updateable after every access, and efficiently and interactively computable by the investigator and the proxy. Thus, one share for the index will be stored encrypted alongside with the encrypted datablock, while the other will be stored on the proxy.

From the above discussion, it becomes evident that along every encrypted block we have to store more information. In particular, we store two encrypted shares and each datablock's `index`. This data structure we will call a 'packet' and we define it in the following.

Definition 21.
Let $\mathcal{E}$ be a semantically secure, re-randomisable, homomorphic encryption scheme with security parameter λ, $(\mathtt{skid}, \mathtt{pkid})$, $(\mathtt{skbf}, \mathtt{pkbf})$, $(\mathtt{skdat}, \mathtt{pkdat})$ *three secret- public- key pairs, and $\mathcal{K}$ a homomorphic hash function such that $\mathcal{K} : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^\lambda$. We call a packet of the datablock with identifier* id*, the tuple*

$$(c_1, c_2, c_3, c_4) =$$

$$\Big(\mathcal{K}(\mathtt{IDshare}_0, \mathtt{IDshare}_1), \mathcal{E}_{\mathtt{pkid}}(\mathtt{IDshare}_1), \mathcal{E}_{\mathtt{pkbf}}(\mathtt{BFshare}_1), \mathcal{E}_{\mathtt{pkdat}}(\mathtt{dat})\Big),$$

where c_1 *is the datablock's* index, c_2 and c_3 are encrypted shares for computing the datablock's index and c_4 is the encrypted data.

5.2. The Protocol in Detail

5.2.1. Initialisation and Authorisation

Initialization and Upload

Before the client uploads his encrypted datablocks to the cloud, he creates an equal amount of fake datablocks and assigns a unique identifier (avoiding collisions) to every datablock (real and fake). The client then initialises the system parameters issuing the shares $\texttt{IDshare}_0$, $\texttt{IDshare}_1$, $\texttt{BFshare}_0$, $\texttt{BFshare}_1$, and creating the encryption scheme's keys: $(\texttt{skid}, \texttt{pkid})$ which is used for encrypting all the identifier shares of the datablocks (the c_2 part of the datablock's packet), $(\texttt{skbf}, \texttt{pkbf})$ which is used for encrypting the packet's Bloom filter share (the c_3 part of the datablock's packet), $(\texttt{skdat}, \texttt{pkdat})$ which is used for encrypting the datablock's data (the c_4 part of the datablock's packet), and $(\texttt{skbf}', \texttt{pkbf}')$ for encrypting the Bloom filter contents. For every datablock (real and fake) the client then creates the datablock's corresponding packet (see Definition 21). The client distributes the keys and the shares to the parties as follows: The cloud receives the secret key $\texttt{skbf}'$, while the proxy receives the shares $\texttt{IDshare}_0$ and $\texttt{BFshare}_0$, as well as the keys skid and skbf. An overview of the keys and shares distributed to the various parties can be found in Table 5.1.

The client then uploads all real and fake packets in random order to the cloud, who stores them on the last level of the pyramid, while all levels above are initialized and left empty. Finally, the client sends the list of the fake ids to the proxy. Once the packets have been uploaded, a Bloom filter creation (see Section 5.3) is initiated between the cloud and the proxy, resulting in the Bloom filter for the last level, which is then stored encrypted on the proxy.

Authorization

At the beginning of an access session in which the investigator wants to read encrypted datablocks from the cloud, the client creates a new temporary key pair $(\texttt{sktemp}, \texttt{pktemp})$ which will function as the access control mechanism. The investigator then receives from the client the authentication token, which consists of the shares $\texttt{IDshare}_0$, $\texttt{IDshare}_1$, $\texttt{BFshare}_1$, and the secret temporary key sktemp. From then on, all messages exchanged between the investigator and the proxy, and between the investigator and the proxy, will be proxy re-encrypted. Observe that even after the decryption, the investigator only sees encrypted data. For ease of exposition however, and without loss of rigor, in the following we will avoid using expressions such as 'a proxy re-encryption of a proxy re-encrypted message', and will always refer to one proxy re-encryption, thus not mentioning again the key pair $(\texttt{sktemp}, \texttt{pktemp})$.

	$\texttt{IDshare}_0$	$\texttt{IDshare}_1$	$\texttt{BFshare}_0$	$\texttt{BFshare}_1$	skid	skbf	$\texttt{skbf}'$
Client	•	•	•	•	•	•	•
Investigator	•	•		•			
Cloud							•
Proxy	•		•		•	•	

Table 5.1: Overview of the values known to all parties. $\texttt{IDshare}_0$, $\texttt{IDshare}_1$: Shares for computing the current index of the datablock, $\texttt{BFshare}_0$, $\texttt{BFshare}_1$: Shares for determining the Bloom filter positions for the datablock; skid: secret key to decrypt a packet's encrypted c_2 part; skbf: secret key to decrypt a packet's encrypted c_3 part; $\texttt{skbf}'$: secret key to decrypt encrypted Bloom filter bits;

5.2.2. Access

Read and Write Operations

Like classical ORAMs, Proxy-ORAM supports two operations, Read and Write. At the end of a Read or Write operation, the investigator puts back on the top level of the cloud's pyramid *two* packets: the real one and a fake one. In case of a Read, the investigator simply re-randomises the elements c_2, c_3, c_4 of the downloaded real packet, while in case of a Write he additionally replaces c_4 of the real packet with an encryption of the new data. Regardless of the operation performed, the packet's index c_1 is updated. Since c_4 is a semantically secure encryption of the datablock's data, and thus a re-randomised ciphertext is indistinguishable from a fresh encryption, we can focus from here on only on the case of a Read operation. Note that, due to this property, neither the proxy nor the cloud can distinguish a Read from a Write.

The pseudocode for the access protocol is given in Protocol 3, the steps of which we describe here. The investigator's query for a datablock with identifier id runs through every ORAM level and works as follows: If the level is empty then there is nothing to be done. If the level is not empty, then the investigator first receives the id of a *fake* packet to be found on that level from the cloud (step 4). This way, the investigator is guaranteed to retrieve a datablock from the cloud. After this initial step, the investigator interactively with the proxy computes the Bloom filter positions corresponding to the datablock he is querying for, assuming that it resides on that level (steps 7 to 13). First, the investigator blinds his share for the Bloom filter, $\texttt{BFshare}_1$ with a randomly chosen value (taking care that for each of the ξ different hash values a different blinding value is chosen) and sends this blinded share to the proxy. The proxy then, using his share for the Bloom filter, creates the Bloom filter positions for the given identifier. These values are then sent by the proxy back to the investigator, who removes the blindings and now knows which positions of the encrypted Bloom filter he should ask for. The investigator then requests the encrypted Bloom filter contents of these positions from the proxy. The investigator now blinds the encrypted Bloom filter contents (each with a different blinding value) and sends them to the

Protocol 3: Proxy-ORAM.Access(id)

define ξ the number of hash functions used for the Bloom filter;
1 $\texttt{blockfound} = 0$;
2 **for** $i = 1$ **to** $\lceil \log_2 N \rceil$ **do**
3 **if** *level* i *is not empty* **then**
4 $\mathsf{id}_{\text{fake}} \leftarrow$ GetFakeIDfromCloud;
5 $\texttt{bf\_flag} = 1$;
6 **for** $j = 1$ **to** ξ **do**
7 $v \xleftarrow{R} \{0,1\}^*$;
8 $\texttt{B\_part}$ = Blind($\texttt{BFshare}_1, v$);
9 Send $\texttt{B\_part}$ to Proxy /* the proxy will now compute all Bloom filter positions */;
 PROXY:
10 $\texttt{B\_full}$ = Combine(Blinded($\texttt{BFshare}_1$,$\texttt{BFshare}_0$);
11 Send $\texttt{B\_full}$ to Investigator/* all positions are now blinded */;
 INVESTIGATOR:
12 B = Unblind($\texttt{B\_full}$);
13 $\texttt{EBF} \leftarrow$ GetEncryptedBFpositionFromProxy(B);
14 $v \xleftarrow{R} \{0,1\}^*$;
15 Blind($\texttt{EBF}$,v);
16 SendToCloud($\texttt{EBF}$);
17 $\texttt{EBF}'$ = GetBlindedDecryptedBFContentsFromCloud;
18 $B \leftarrow$ Unblind($\texttt{EBF}'$);
19 **if** $B == 0$ **then**
20 $\texttt{bf\_flag} = 0$;
21 **if** $\texttt{bf\_flag} == 0$ **or** $\texttt{blockfound} == 1$ **then**
22 $\texttt{RetBlock}$ = GetFromCloud($\texttt{index}(\mathsf{id}_{\text{fake}})$);
23 **else**
24 $\texttt{RetBlock}$ = GetFromCloud($\texttt{index}(\mathsf{id})$);
25 $(\perp, c_2^f, c_3^f, c_4^f)$ = CreateFakepacket($\mathsf{packet}(\mathsf{id}_{\text{fake}}^{\texttt{lastlevel}})$);
26 $(\perp, c_2^r, c_3^r, c_4^r)$ = Rerandomise($\mathsf{packet}(\mathsf{id})$);
 /* the indices c_1^r and c_1^f will be built in the next reshuffle between the cloud and the proxy */;
27 Send $\mathsf{id}_{\text{fake}}^{\texttt{lastlevel}}$ to cloud;
28 UploadToCloudInRandomOrder($(\perp, c_2^f, c_3^f, c_4^f), (\perp, c_2^r, c_3^r, c_4^r)$);
 CLOUD:
29 Mark all $\texttt{fakeids}$ as 'used';
30 Mark $\mathsf{id}_{\text{fake}}$ received from investigator as 'unused';
31 **if** $\texttt{level}_1$ has enough space **then**
32 Create $\texttt{level}_0$;
33 ORAM.Reshuffle($\texttt{level}_0$,$\texttt{level}_1$);
34 **else**
35 ORAM.Reshuffle($\texttt{level}_1$,$\texttt{level}_2$);

cloud (steps 13- 16). The latter decrypts the blinded and encrypted Bloom filter contents, and sends them back to the investigator. The investigator now removes the blindings and learns if the datablock is on that level or not (steps 19 and 20). If the datablock is not on the level or it has been found on a previous level, then the investigator retrieves from the cloud the fake datablock from this level *which is certainly there*, otherwise, the investigator retrieves the real datablock he was looking for (step 24). Once the datablock has been found, the investigator continues to access the remaining levels of the pyramid, this time however asking in the last step directly for the fake datablock. Observe here, multiple versions of the same datablock are expected to be found in various levels of the structure, however the newest version will always be found on the level closest to the top of the pyramid.

On completion of accessing every pyramidal level, the investigator generates a fake packet by computing the elements c_2, c_3, for the $\mathsf{id}_{\text{fake}}$ retrieved from the *last ORAM level* and forming c_4 by encrypting 'fake' data like for example '0^B' (step 25). The investigator then re-randomises the elements c_2, c_3 and c_4 of the real packet he retrieved and sends the $\mathsf{id}_{\text{fake}}$ he got from the last level to the cloud (steps 25 to 28). The cloud stores $\mathsf{id}_{\text{fake}}$ on a list of fake identifiers that are available on the first level and marks all the fake ids that he gave throughout the query as 'used', so that they are not re-used in subsequent queries (steps 29 to 30). Finally the investigator sends the real and the fake packets in random order to the cloud. The cloud creates the indices for the two packets interactively with the proxy in the following way: If the cloud does not have enough space to hold the two elements in the first level, then a `Reshuffle` is triggered, during which the freshly added elements and the ones that were already in the first level are shuffled, merged with the first level and their indices are updated. Otherwise, if there is enough space for the two elements to be stored in the cloud's first level, the two packets to be added are considered as a 'level 0' and are reshuffled with the elements of the first level (steps 31 to 35). Thus the indices of the two packets are generated, their order is randomly permuted and a new Bloom filter for 'level 1' is created. These operations are described in detail in the 'Reshuffle' operation, which is run between the cloud and the proxy, and which we describe in detail in the following.

5.2.3. Reshuffle

Once a level is entirely filled with packets, it is emptied recursively to the next one, until a level is reached, which does not overflow after its previous level is emptied into it. In order to maintain access pattern hiding, the datablocks have to be obliviously reshuffled while they are pushed into the new level. Typically in hierarchical ORAM constructions [GO96; WSC08; Goo+12b; Goo+12a; WST12] the oblivious reshuffle phase is performed by means of oblivious sorting, which can be performed between the client and the server by using sorting networks, or other efficient sorting techniques [Goo14; Goo11a; Goo11b] that require very little memory usage from the client. In our scenario however, we do not want to involve the client (or the even the investigator) at all during this communication intensive procedure – in fact, as we have seen, the only client involvement is handing over to the investigator the authorization token. We achieve this goal by splitting our `Reshuffle` into two distinct phases that run solely between the proxy and the cloud: First, the Bloom filter creation phase, where the Bloom filter for the deepest level of the

Protocol 4: Proxy-ORAM.Reshuffle.BFCreation($\texttt{level}_{l-1}, \texttt{level}_l$)

define λ the security parameter ;
define ξ the number of hash functions used for the Bloom filter;
define $\texttt{bfsize} = b(\lambda, |\texttt{level}_l|)$;
PROXY:
Discard Bloom filter of $\texttt{level}_{l-1}$;
Discard Bloom filter of $\texttt{level}_l$;
for every key **in** Bloom filter keys **do**
    Discard key;
    key $\leftarrow \{0,1\}^{\lambda}$;
CLOUD:
for every $\mathsf{id}_{\text{fake}}$ **in** $\texttt{level}_{l-1}$ **do**
    Mark $\mathsf{id}_{\text{fake}}$ 'not used';
    Move $\mathsf{id}_{\text{fake}}$ to $\texttt{level}_l$;
$L \leftarrow \texttt{level}_{l-1}$;
$L \leftarrow \texttt{level}_l$;
$L' \leftarrow \emptyset$;
$(\texttt{tsk}, \texttt{tpk}) \leftarrow \texttt{KeyGen}(1^{\lambda})$ /* key pair for proxy re-encryption */;
for every packet **in** L **do**
    $c_3' \leftarrow \texttt{ProxyReEncrypt}_{\texttt{tpk}}(\text{packet}.c_3)$;
    $L' \leftarrow c_3'$ /* now L' contains all $\texttt{BFshare}_1$ proxy re-encrypted */;
Send L' and tpk to Proxy;
PROXY:
$L^* \leftarrow \emptyset$;
for every Share **in** L' **do**
    $\texttt{bfposition} = \texttt{COMBINE}(\texttt{Share}, \texttt{BFshare}_0)$;
    $\texttt{bfposition} = \texttt{Dec}_{\texttt{skbf}}(\texttt{bfposition})$ /* now cloud can decrypt */;
    $L^* \leftarrow \texttt{bfposition}$;
Send L^* and tpk to Cloud;
CLOUD:
$\mathsf{BLOOMFILTER}_l[\texttt{bfsize}]$;
for every i **in** $\mathsf{BLOOMFILTER}_l$ **do**
    $i = \texttt{Enc}_{\texttt{pkbf}'}(0)$;
for every bfposition **in** L^* **do**
    $\texttt{bfposition} = \texttt{Dec}_{\texttt{tsk}}(\texttt{bfposition})$;
    $\mathsf{BLOOMFILTER}_l[\texttt{bfposition}] = \texttt{Enc}_{\texttt{pkbf}'}(1)$;
Send $\mathsf{BLOOMFILTER}_l$ to Proxy;

two levels being reshuffled is created, and then the update phase, during which the packets of the two levels are merged, leaving the smaller of the levels empty. In details the reshuffle operation from level $l-1$ into level l works as follows:

Phase 1: Bloom filter Creation.

In the first reshuffle phase, the current Bloom filters of levels $l-1$ and l are destroyed and a new Bloom filter for level l is created. The operation is interactively performed between the proxy and the cloud, who compute the positions of the Bloom filter that need to be set to '1', without being able to identify the datablocks in the two levels. We describe the protocol here, and for more details we refer to the protocol's pseudocode, illustrated in Protocol 4.

In the first step, the proxy destroys the Bloom filters of both levels l and $l-1$, as well as the keys used for computing the Bloom filters. Then, the proxy creates new random keys for the Bloom filter of level l (lines 1 to 5). In the next steps, the cloud marks all the fake identifiers from level $l-1$ as 'not used' and moves them to the available fake identifiers of level l (lines 6 to 8).

The cloud now creates a list that contains all the packets from levels $l-1$ and l. Furthermore, he creates a temporary pair of random keys $(\mathtt{tsk}, \mathtt{tpk})$ for the encryption scheme used to encrypt the packets' c_3 part. This key pair will be used to proxy re-encrypt c_3 of all packets to be reshuffled in the following. For every packet (c_1, c_2, c_3, c_4) present on the list, the cloud uses proxy re-encryption to transform $c_3 = \mathtt{Enc}_{\mathtt{pkbf}}(\mathtt{BFshare}_1)$ into an encryption under the public key $\mathtt{tpk} + \mathtt{pkbf}$ (lines 13 to 15), and sends the resulting encrypted list and the key $\mathtt{tpk}$ to the proxy.

The proxy can now compute the Bloom filter positions that will be set to '1' in the new Bloom filter for level l. For every packet to be inserted in level l, the proxy computes the ξ Bloom filter positions, using the share $\mathtt{BFshare}_0$ which is known to him. Taking advantage of the proxy re-encryption property, the proxy transforms the encryption into an encryption under the public key $\mathtt{tpk}$ by removing the $\mathtt{skbf}$ dependency (lines 18 to 21). The proxy finally sends the resulting list of elements back to the cloud. Observe that this list now contains all the Bloom filter positions that must be set to '1', and each of them can now be decrypted by the cloud since the latter knows $\mathtt{tsk}$.

The cloud initialises a Bloom filter of size $\mathtt{bfsize}$, with all values set to $\mathtt{Enc}_{\mathtt{pkbf}'}(0)$. For every element of the received list, the cloud decrypts the elements received in the previous step, and sets the Bloom filter positions to a 'fresh' encryption of '1', $\mathtt{Enc}_{\mathtt{pkbf}'}(1)$ (steps 26 to 28). Finally, the cloud sends the resulting encrypted Bloom Filter to the proxy and discards the key pair $(\mathtt{tsk}, \mathtt{tpk})$.

Phase 2: Update.

In this second phase, the packets stored on levels $l-1$ and l are merged into level l. All the packets are re-randomised and randomly permuted, while a new random index is computed for each one of them. This procedure is illustrated in Protocol 5. Firstly, all packets currently stored on level $l-1$ are moved to level l. Then, the cloud creates a new pair of temporary proxy re-encryption keys $(\mathtt{tsk}, \mathtt{tpk})$. For every

Protocol 5: Proxy-ORAM.Reshuffle.Update($\texttt{level}_{l-1}, \texttt{level}_l$)

CLOUD:

All packets from $\texttt{level}_{l-1}$ are moved to $\texttt{level}_l$;

$(\texttt{tsk}, \texttt{tpk}) \leftarrow \texttt{KeyGen}(1^\lambda)$;

for every packet **in** $\texttt{level}_l$ **do**

  $\text{packet}.c_1 = \perp$ /* remove previous c_1 */;

  $\text{packet}.c_2 = \texttt{ProxyReEncrypt}_{\texttt{tpk}}(\text{packet}.c_2)$;

  $\text{packet}.c_3 = \texttt{ProxyReEncrypt}_{\texttt{tpk}}(\text{packet}.c_3)$;

  $\text{packet}.c_4 = \texttt{ProxyReEncrypt}_{\texttt{tpk}}(\text{packet}.c_4)$;

Send $\texttt{level}_l$ and tpk to the proxy;

PROXY:

for every packet **in** $\texttt{level}_l$ **do**

  $\text{packet}.c_1 = \texttt{COMBINE}(\text{packet}.c_2, \texttt{IDshare}_0)$;

  $\text{packet}.c_1 = \texttt{Dec}_{\texttt{skid}}(\text{packet}.c_1)$ /* remove skid dependency */;

  $\text{packet}.c_1 = \texttt{ReRandomize}(\text{packet}.c_1)$;

  $\text{packet}.c_2 = \texttt{ReRandomize}(\text{packet}.c_2)$;

  $\text{packet}.c_3 = \texttt{ReRandomize}(\text{packet}.c_3)$;

  $\text{packet}.c_4 = \texttt{ReRandomize}(\text{packet}.c_4)$;

$\texttt{level}_l = \texttt{RandomlyPermute}(\texttt{level}_l)$;

Send $\texttt{level}_l$ to the cloud;

CLOUD:

for every packet **in** $\texttt{level}_l$ **do**

  $\text{packet}.c_1 = \texttt{Dec}_{\texttt{tsk}}(\text{packet}.c_1)$/* results in new random plaintext index */;

  $\text{packet}.c_2 = \texttt{Dec}_{\texttt{tsk}}(\text{packet}.c_2)$/* remains encrypted under pkid */;

  $\text{packet}.c_3 = \texttt{Dec}_{\texttt{tsk}}(\text{packet}.c_3)$/* remains encrypted under pkbf */;

  $\text{packet}.c_4 = \texttt{Dec}_{\texttt{tsk}}(\text{packet}.c_4)$/* remains encrypted under pk */;

packet stored on this newly merged level, the cloud first removes c_1 and transforms the parts (c_2, c_3, c_4) into encryptions under the public key `tpk`. The cloud then sends the transformed parts (c_2, c_3, c_4) of each packet to the server, along with the public keys `tpk` to the proxy (steps 3 to 8).

For every packet in the received level, the proxy uses his share $\texttt{IDshare}_0$ with the packet's encrypted share c_2, in order to create the packet's new index c_1. The newly created packet index is now encrypted under `tpk`. Using `skid` as a re-encryption key, the proxy removes this key's dependency from each packet's c_1 This results in a c_1 that can be decrypted by the cloud. Finally, the proxy re-randomises all obtained elements (c_1, c_2, c_3, c_4) for every packet, permutes the updated packet list, and sends this updated and permuted list back to the cloud (steps 9 to 17).

The cloud now decrypts the index c_1 of every received packet using the secret key `tsk` and removes the `tsk` dependency from the parts (c_2, c_3, c_4) from all the packets. The result of this process is a randomly permuted list of packets with fresh indices.

5.3. Analysis

For Proxy-ORAM to fulfill the minimal soundness conditions from Definition 12, two key factors have to hold. For one, a false positive must occur in the Bloom filter only with very small probability, otherwise the investigator will retrieve another datablock than the one he was asking for. Since, as we will see, Bloom filter false positives pose also an important security threat, it suffices to adjust the Bloom filter size, so that the probability of a false positive occurrence is negligible in the security parameter.

Of equal importance for the soundness of Proxy-ORAM is the existence of fake datablocks in every level. This is achieved by having the fake identifiers stored on the cloud, and marking them as 'used' and 'unused'. The question that arises, however, is if there are always enough fake identifiers available for every level. We prove that this is always the case:

Proposition 3.
In every query there are always enough fake ids for the proxy to provide from every level.

Proof. (*sketch*) The proof is done by an induction in the number of fake datablocks available in every level. Observe that during the initialisation, all real and (the equally many) fake datablocks along with the fake identifiers are stored on the last level of the cloud. Thus, during the investigator's first access, a fake identifier (and its respective datablock) is guaranteed to be found on that level. Assume now that during the n-th access, the investigator found a fake identifier in every level. Observe that as soon as a level gets full (and thus has also exhausted all its available fake identifiers), a reshuffle for that level and the subsequent level is triggered, leaving the second of the reshuffled levels with fresh fake identifiers, and the first reshuffled level empty as detailed in Protocol 4. Recalling that during every access, only the

non-empty levels are considered, we have that in the $n+1$-th access, the investigator will be able to find a fake identifier in every level, that he will examine. □

Proposition 4 (Proxy-ORAM Correctness)**.**
If the probability of a false positive occurrence in the Bloom filter is negligible in the security parameter λ, Proxy-ORAM is sound according to Definition 12, i.e., for every query, the ORAM server (cloud) returns only the correct datablock, except with negligible probability.

Proof. (*sketch*) For a query of the packet with identifier id, the investigator goes through all ORAM levels and queries each one's Bloom filter. If the packet resides on level l, then the Bloom filter indicates it and the investigator retrieves from the cloud the packet with identifier $\mathtt{index}(\mathsf{id})$, except with negligible probability, which occurs in the case of a Bloom filter false positive. Otherwise he retrieves a fake element by asking for the packet with the fake id $\mathsf{id}_{\text{fake}}$ (retrieved from the cloud), which is guaranteed to reside on the level. After the investigator has retrieved the packet he wanted, he asks only for fake packets. Thus, after going through all ORAM's levels, the investigator retrieves the packet he was querying for and this datablock only, as the other datablocks are fake datablocks. □

Proposition 5.
Proxy-ORAM storing N real datablocks and B datablocks on its first level, has amortised communication and computation cost of $O(\log N)$ for the cloud and proxy.

Proof. By Proxy-ORAM's description, we have that reshuffling levels $i-1$ and i involves the exchange of $O(2^i)$ datablocks between the cloud to the proxy. As we have seen, reshuffle is a recursive procedure, and thus it can potentially trigger reshuffles in higher levels. Let $C(i,j)$ denote the total amount of datablocks moved and computed upon by the proxy (or the cloud) at level i during the j-the access. Suppose that the first ORAM level can hold B datablocks, and observe that between j and $2j$ accesses, level i (for some $i \leq \lceil \log_2 2j \rceil$) will have been reshuffled $2^{\lceil \log_2 2j \rceil - i}$ times, out of which $2^{\lceil \log_2 2j \rceil - i - 1}$ level i will be half-full (i.e., will have B^{i-1} datablocks) and $2^{\lceil \log_2 2j \rceil - i - 1}$ level i will be full (i.e., will have B^i datablocks). Thus, after N accesses the total amount of datablocks communicated and computed upon between the proxy and the cloud will be

$$T(N) = \sum_{i=1}^{\log_2 N} \sum_{j=1}^{\log_2 N} C(i,j) = \sum_{i=1}^{\log_2 N} \sum_{j=1}^{\log_2 N} 2^{\lceil \log_2 2j \rceil - i - 1} \left(\frac{B+1}{2B}\right) 4^j \left(\frac{B}{2}\right)^i \leq$$

$$\left(\frac{B+1}{2B}\right) \log_2 N \sum_{j=1}^{\log_2 N} B^i \leq B^{2+\log_2 N} \log_2 N \leq cN \log_2 N \text{ for some } c > 1,$$

which means that $T(N) \in O(N \log_2 N)$, and thus the amortised cost for accessing one block is in $O(\log_2 N)$. □

5.4. Security

Before proving Proxy-ORAM's security formally, we give the intuition of the fact that Proxy-ORAM provides access pattern hiding, assuming that the proxy and the cloud are semi-honest and do not collude.

The basic idea behind Proxy-ORAM's security has to do with the fact that the various components are shared between non-colluding parties. The proxy holds the encrypted Bloom filters, but cannot decrypt them. Although he sees the positions of the Bloom filters accessed, this does not give him any information, under the assumption that cryptographically secure hash functions are used for building the Bloom filter, that these keys are renewed in every reshuffle, and every datablock (real or fake) in every level is accessed only once before the level is reshuffled, Observe further that, although the proxy has both `skid` and `skbf` which would allow him to decrypt c_2 and c_3 of every packet respectively, during the course of all protocols, the proxy only sees these elements proxy re-encrypted, thus losing his ability to decrypt them.

As far as the cloud is concerned, observe that the knowledge of the secret key $\texttt{skbf}'$ used to decrypt the Bloom filter contents does not jeopardise the system's security. Indeed, during the access, the cloud sees only blinded Bloom filter contents that he has to decrypt. Furthermore, although during the Bloom filter creation phase the cloud sees the plaintext Bloom filter contents, he cannot correlate the ids to positions, since he does not have any shares for this. Finally, the cloud sees which fake id could have potentially been asked from the investigator, but again this does not give him any information, because the cloud cannot create the respective fake packet (and thus the fake index c_1).

The existence of the Bloom filter, however, can potentially be a source of insecurity (as also pointed out in [KLO12]). Take for example the following case: Suppose that a Bloom filter false positive occurs, and hence the investigator retrieved an incorrect datablock from level i. Then, it might happen that in his next query, the investigator will ask for exactly the same Bloom filter positions in level i, something which would have never been the case if the correct datablock had been previously found. In fact the datablock would have been now found on the first level, and at level i the investigator would ask for a fake datablock. Clearly, both the proxy and the cloud will notice this immediately, and thus the access pattern privacy of the architecture would have been broken. Thus, the Bloom filters have to be large enough, so that the probability of a false positive is negligible in the security parameter.

With the above observations in mind, we can now proceed with proving Proxy-ORAM's security more formally. For an architecture such as Proxy-ORAM, which is based on a variety of cryptographic primitives, proving its security formally would have been a very challenging task. Using our new framework from Chapter 4 however, the proof of security is done in a very simple and straightforward way. First, we identify all the cryptographic components used in the architecture, and then bind them all together in the IND-CQA game by further defining each component's security in a game-based manner.

We summarise and recall here the various security guarantees that we need for the security of Proxy-ORAM. Firstly, the encryption scheme used in Proxy-ORAM must be re-randomisable by knowledge of

the public key. This is a property achieved by most asymmetric homomorphic encryptions schemes, and is vital in Proxy-ORAM's security since the cloud and the proxy see only encrypted messages which have to be re-randomised in order to destroy any correlation between them, as they are moved from one level to the other during the reshuffle procedure.

The encryption scheme used in Proxy-ORAM must also provide indistinguishability of blinded encryptions. Clearly this property is of grave importance, since in the course of the protocols, many encrypted messages (such as the Bloom filter contents during the access protocol) are blinded, and then encrypted. We model this property by means of the following game.

Definition 22 ($\mathsf{Game}_{\mathtt{A},\mathcal{E}}^{\mathtt{IND\text{-}CBA}}(\lambda)$)**.**
Let $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Blind})$ *be a semantically secure homomorphic encryption that allows blinding of ciphertexts,* λ *a security parameter and* $\mathtt{A}$ *a PPT adversary for the encryption scheme. The computational indistinguishability of blinded ciphertexts game under adaptive chosen message attack,* $\mathsf{Game}_{\mathtt{A},\mathcal{E}}^{\mathtt{IND\text{-}CBA}}(\lambda)$*, is played between* $\mathtt{A}$ *and a challenger* $\mathtt{C}$ *and proceeds as follows:*

1. $\mathtt{C}$ *generates the key pair* $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathcal{E}.\mathtt{KeyGen}(\lambda)$*, and hands the public key* $\mathtt{pk}$ *to* $\mathtt{A}$*;*
2. *First learning phase: for* q_1 *times (polynomially many times in* λ*),* $\mathtt{A}$ *selects (adaptively) a message* m_i*, encrypts it using* $\mathtt{pk}$*, issues* c_i *and asks a blinding oracle* $\mathcal{O}$ *for a blinding of* c_i*;*
3. *Challenge phase:* $\mathtt{A}$ *issues two ciphertexts* (c_0, c_1)*;*
4. $\mathtt{C}$ *flips a random secret bit* $b \xleftarrow{\$} \{0,1\}$*, and blinds* c_b
5. $\mathtt{A}$ *receives the blinded encryption* c_b*;*
6. *Second learning phase: Similarly to the first learning phase, for* q_2 *times (polynomially many in the security parameter)* λ*,* $\mathtt{A}$ *selects (adaptively) a message* m_i*, encrypts it using* $\mathtt{pk}$*, issues* c_i *and asks* $\mathcal{O}$ *for a blinding of* c_i*;*
7. *At the end,* $\mathtt{A}$ *outputs a bit* b'*;*

$\mathtt{A}$ *wins the game if and only if* $b = b'$*.*

Definition 23 (Indistinguishability Under Adaptive Chosen Blinded Attack)**.**
A public key encryption scheme $\mathcal{E}$ *has computationally indistinguishable blinded ciphertexts under adaptive chosen query attack (or, it is IND-CBA-secure) if and only if for any PPT adversary* $\mathtt{A}$ *the following holds:*

$$\left| \mathrm{Prob}\left[\mathsf{Game}_{\mathtt{A},\mathcal{E}}^{\mathtt{IND\text{-}CBA}}(\lambda) = 1 \right] - \frac{1}{2} \right| \leq \mathit{negl}(\lambda).$$

In many steps of the Proxy-ORAM protocol, an encrypted message has to be proxy re-encrypted. As we have seen in Section 3.2.1, the core idea of this procedure is to produce a new temporary public-, private-key pair, and using the new private key in order to alter the public key under which a given ciphertext is originally encrypted. The original scheme from [BBS98] was further improved in [Ate+06], where

(among others) the non-collusion requirement was removed. However since we already assume non-collusion between the proxy and the cloud, the basic ideas from [BBS98] suffice. In Proxy-ORAM, proxy re-encryption is needed in various steps, such as the authorisation step at the very beginning, or during the Bloom filter creation. In all instances that proxy re-encryption appears in the protocol, it is coupled with a re-randomisation of the ciphertext. Thus, proving the security of the protocol must take this into account. We formalise this, by means of the following game.

Definition 24 ($\mathsf{Game}^{\texttt{PREN-IND-CPA}}_{\mathtt{A},\mathcal{E}}(\lambda)$)**.**
Let $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{PREN})$ *be a semantically secure homomorphic encryption that allows proxy re-encryption of ciphertexts,* λ *a security parameter and* $\mathtt{A}$ *a PPT adversary for the encryption scheme. The computational indistinguishability of proxy re-encrypted ciphertexts game under adaptive chosen message attack,* $\mathsf{Game}^{\texttt{PREN-IND-CPA}}_{\mathtt{A},\mathcal{E}}(\lambda)$*, is played between* $\mathtt{A}$ *and a challenger* $\mathtt{C}$ *and proceeds as follows:*

1. $\mathtt{C}$ *generates two key pairs:* $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathcal{E}.\mathtt{KeyGen}(\lambda)$*, and of which* $(\mathtt{tpk}, \mathtt{tsk}) \leftarrow \mathcal{E}.\mathtt{KeyGen}(\lambda)$*, and hands over to* $\mathtt{A}$ $\mathtt{pk}$ *and* $\mathtt{tpk}$*. The key* $\mathtt{pk}$ *will be used for encrypting messages, and* $\mathtt{tpk}$ *will be used for proxy re-encrypting ciphertexts;*
2. *First learning phase: for* q_1 *times (polynomially many times in* λ*),* $\mathtt{A}$ *selects (adaptively) a message* m_i*, encrypts it using* $\mathtt{pk}$*, issues* c_i *and asks a proxy re-encryption oracle* $\mathcal{O}$ *to proxy re-encrypt* c_i*;*
3. *Challenge phase:* $\mathtt{A}$ *issues two ciphertexts* (c_0, c_1)*;*
4. $\mathtt{C}$ *flips a random secret bit* $b \xleftarrow{\$} \{0,1\}$*, and using* $\mathtt{tpk}$ *proxy re-encrypts* c_b*, and re-randomises it, yielding* c_b^**;*
5. $\mathtt{A}$ *receives the proxy re-encryption of* c_b^**;*
6. *Second learning phase: for* q_2 *times (polynomially many times in* λ*),* $\mathtt{A}$ *selects (adaptively) a message* m_i*, encrypts it using* $\mathtt{pk}$*, issues* c_i *and asks* $\mathcal{O}$ *for a proxy re-encryption of* c_i*;*
7. $\mathtt{A}$ *outputs a bit* b'*.*

$\mathtt{A}$ *wins the game if and only if* $b = b'$*.*

Definition 25 (Proxy Re-encryption Ciphertext Indistinguishability Under Adaptive Chosen Message Attack)**.**
A public key encryption scheme $\mathcal{E}$ *has computationally indistinguishable proxy re-encrypted ciphertexts under adaptive chosen message attack (or, it is* PREN-IND-CPA-secure) if and only if for any PPT adversary $\mathtt{A}$:

$$\left| \mathrm{Prob}\left[\mathsf{Game}^{\text{PREN-IND-CPA}}_{\mathtt{A},\mathcal{E}}(\lambda) = 1 \right] - \frac{1}{2} \right| \leq \mathrm{negl}(\lambda).$$

Proposition 6 (Proxy-ORAM security)**.**
Assume that the proxy and the cloud are semi-honest and not colluding, and that a Bloom filter is used for which a false positive occurs with probability negligible in λ*. Assume further that the public key encryption scheme used is semantically secure, has computationally indistinguishable blinded ciphertexts under*

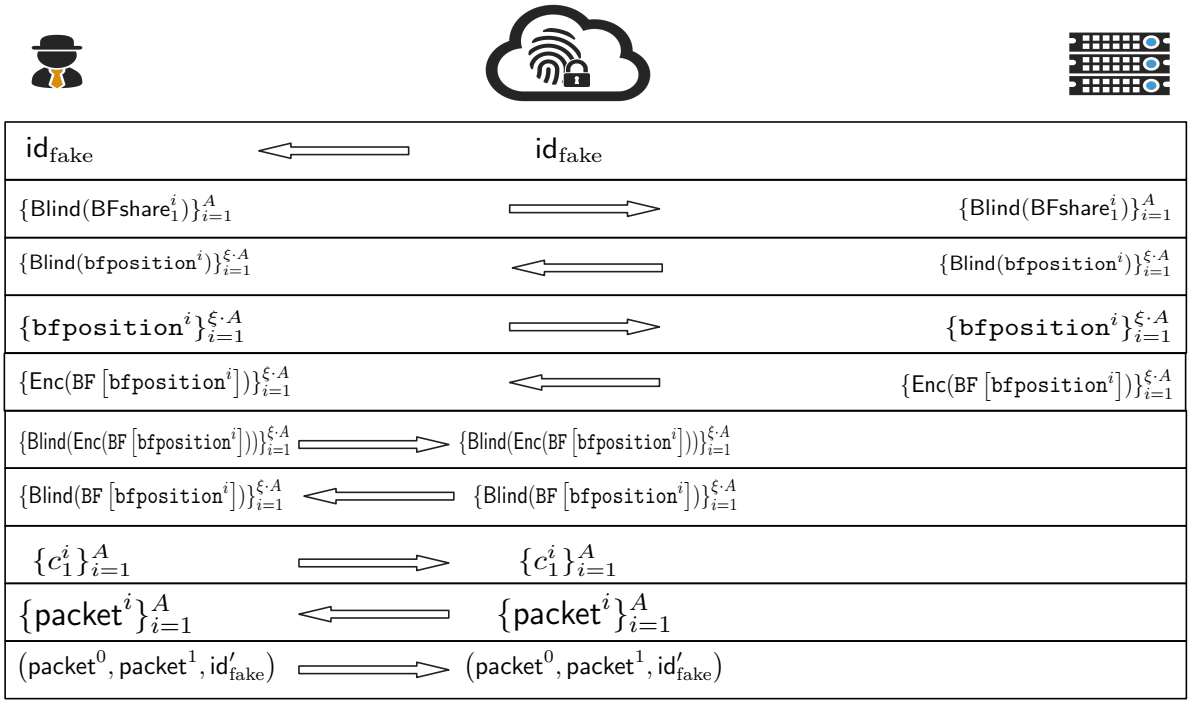


Figure 5.2: Overview of the information exchanged between all parties during the access protocol, with $A = \log N$

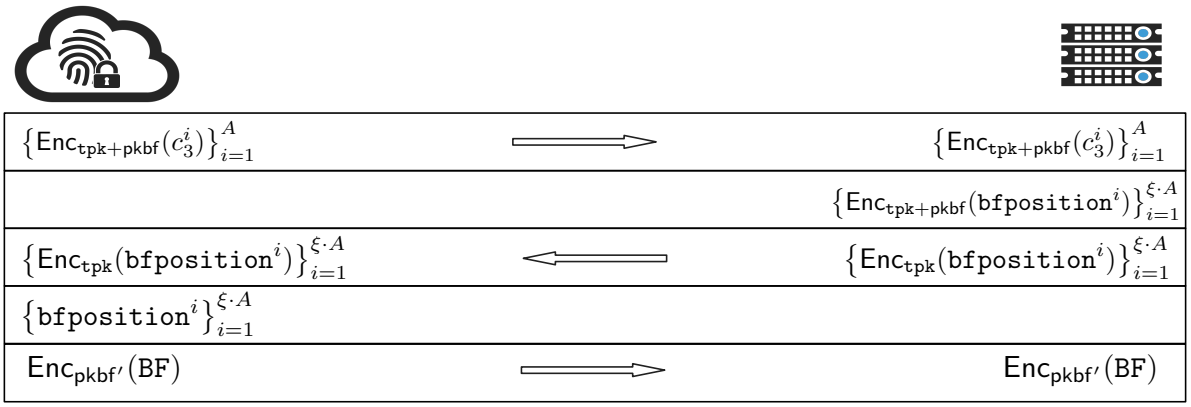


Figure 5.3: Overview of the Bloom filter creation for levels $i-1$ and i, with $A = \frac{3}{2}2^l$

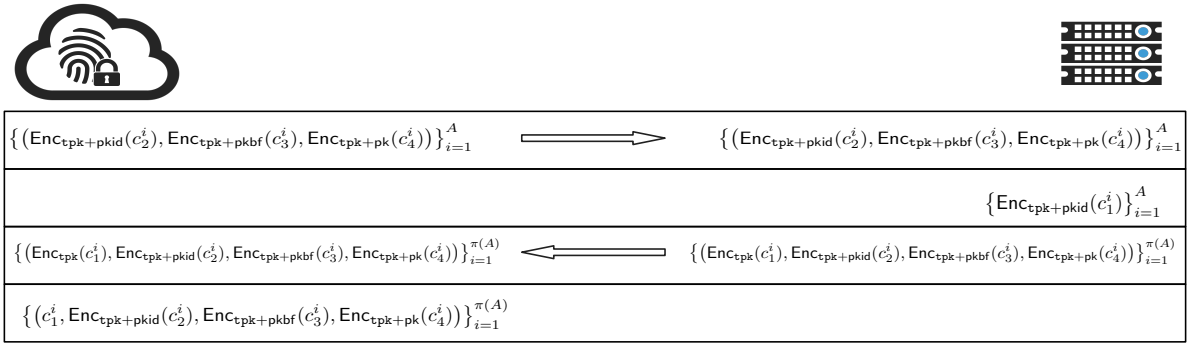


Figure 5.4: Overview of the level update of levels $i-1$ and i, with $A = \frac{3}{2}2^l$

adaptive chosen query attack, and computationally indistinguishable proxy re-encrypted ciphertexts under adaptive chosen plaintext attack. Then Proxy-ORAM is IND-CQA secure according to Definition 15.

Proof. Since the proxy and the cloud do not collude, we can examine each of them separately, every time making use of Definition 15 which considers only one adversarial server. First, assume that there exists a PPT adversary $\mathtt{A}$ that having control of the cloud, wins the IND-CQA game (according to Definition 15) with a non-negligible advantage δ. We will construct a PPT algorithm $\mathtt{B}$ that breaks either the semantic security, or the blinding security, or the re-encryption security of the encryption scheme. Let the cloud's view be the union of the contents of the proxy's and the investigator's, and the proxy's and the cloud's communication tapes (as in Definition 30), and recall that the cloud's view consists of the view during the access and the view during a potential Bloom filter creation and level update. These can be of different lengths, depending on the number of the Proxy-ORAM access we are making (for example, in the first access no reshuffle is triggered) and for a summary of the partys' views, we refer to Figures 5.2 to 5.4. Observe however that in *every* access the cloud sees at least one blinded encryption of the contents of the Bloom filter, at least one encrypted message, and at least one proxy re-encrypted message. This observation will allow us use all of $\mathtt{A}$'s capabilities.

$\mathtt{B}$ plays against three challengers: A $\mathtt{BLIND\text{-}IND\text{-}CBA}$ challenger $\mathtt{C}_0$, a $\mathtt{PREN\text{-}IND\text{-}CPA}$ challenger $\mathtt{C}_1$ and an $\mathtt{IND\text{-}CPA}$ challenger $\mathtt{C}_2$. $\mathtt{B}$ also has access to the respective oracles, $\mathcal{O}_0$, $\mathcal{O}_1$ and $\mathcal{O}_2$ from which he can ask and receive blindings, proxy re-encrypted messages and encrypted messages respectively. $\mathtt{B}$ plays the role of a Proxy-ORAM client, and thus creates a plaintext version of the Proxy-ORAM structure (i.e., both the proxy and the server) that stores N datablocks. $\mathtt{B}$ now encrypts all the structures, by using $\mathcal{O}_2$, and keeps two versions of the Proxy-ORAM: the plaintext version, and the encrypted version of all structures. $\mathtt{B}$ now starts $\mathtt{A}$, who asks for the views of the Proxy-ORAM's cloud for data request sequences he poses. In the pre-challenge phase, $\mathtt{B}$ is able to perfectly simulate both the client and the server of Proxy-ORAM by preparing the corresponding view for a data request sequence $(\mathsf{op}, \mathsf{id}, \mathtt{dat})$, as follows:

1. For every blinding he needs, he asks $\mathcal{O}_0$.
2. Since $\mathtt{B}$ has both $\mathtt{BFshare}_0$ and $\mathtt{BFshare}_1$, he can find which positions of the Bloom filter he has to retrieve. He uses the encrypted Bloom filter to add these to the view.
3. If a reshuffle is triggered, $\mathtt{B}$ runs it in the plaintext view, and simulates it in the encrypted version, using $\mathcal{O}_1$ in order to create the proxy re-encrypted versions when needed.
4. Every time $\mathtt{B}$ needs to perform a re-randomisation of an encrypted packet, he uses $\mathcal{O}_2$.

In the challenge phase, $\mathtt{A}$ passes to $\mathtt{B}$ the tuple $(\mathtt{DRS}_0, \mathtt{DRS}_1)$. $\mathtt{B}$ now randomly selects against which of the three challengers he wants to play. Further, $\mathtt{B}$ picks a random bit b^*, makes two copies of the plaintext version of the structures, and runs in the first copy $\mathtt{DRS}_0$ and in the second $\mathtt{DRS}_1$. $\mathtt{B}$ now forms $\mathtt{DRS}_{b^*}$ as in the pre-challenge phase with the following difference: If he chose to play against $\mathtt{C}_0$, he picks the first position of the plaintext view, where $\mathtt{View}(\mathtt{DRS}_0)$ and $\mathtt{View}(\mathtt{DRS}_1)$ differ and need to be blinded, and forms the challenge for $\mathtt{C}_0$ by sending all the respective encrypted messages $\{c_0^j\}_{j=1}^{\nu}$ and $\{c_1^j\}_{j=1}^{\nu}$. He then picks random challenges to send to $\mathtt{C}_1$ and $\mathtt{C}_2$. If $\mathtt{B}$ chose to play against $\mathtt{C}_1$, he picks the first position of

the plaintext view, where $\texttt{View}(\texttt{DRS}_0)$ and $\texttt{View}(\texttt{DRS}_1)$ differ and need to be proxy re-encrypted, and this way, he forms the challenge for $\texttt{C}_1$ in a similar way as before, while picking random challenges to send to $\texttt{C}_0$ and $\texttt{C}_2$. Finally, if $\texttt{B}$ chose to play against $\texttt{C}_2$, he picks the first position of the plaintext view, where $\texttt{View}(\texttt{DRS}_0)$ and $\texttt{View}(\texttt{DRS}_1)$ differ and need to be re-encrypted, and this way, he forms the challenge for $\texttt{C}_2$ as before, while picking random challenges to send to $\texttt{C}_0$ and $\texttt{C}_1$. The challengers respond, and $\texttt{B}$ injects $\texttt{C}_0$'s response in $\texttt{View}(\texttt{DRS}_{b^*})$ if he chose to play against him, and discards the other responses. Similarly, $\texttt{B}$ injects $\texttt{C}_1$'s response in $\texttt{View}(\texttt{DRS}_{b^*})$ if he chose to play against $\texttt{C}_1$, and discards the other responses, and injects $\texttt{C}_2$'s response in $\texttt{View}(\texttt{DRS}_{b^*})$ if he chose to play against him, discarding the other responses. The rest of $\texttt{View}(\texttt{DRS}_{b^*})$, $\texttt{B}$ creates as in the pre-challenge phase. $\texttt{A}$ now continues with the post-challenge phase, which is ran as the pre-challenge. At some point, $\texttt{A}$ returns a bit b'. $\texttt{B}$ then outputs (b', d_0, d_1) if he chose to play against $\texttt{C}_0$, (d_0, b', d_1) if he chose to play against $\texttt{C}_1$, and (d_0, d_1, b') if he chose to play against $\texttt{C}_2$, where d_0 and d_1 are random bits.

Observe now, that if $b^* = b$, then $\texttt{A}$ gets a view for which his advantage of winning the IND-CQA game applies – we will call this, the *correct* view. Assume further, that each time $\texttt{A}$ gets an incorrect view he outputs 0 with some probability α and 1 with probability $1-\alpha$. Calculating $\texttt{B}$'s total winning probability, we get

$$\text{Prob}[\texttt{B wins } \texttt{C}_0 \text{ or } \texttt{C}_1 \text{ or } \texttt{C}_2] = \frac{1}{3} \cdot \frac{1}{4} \left(3 \cdot \left(\frac{1}{2} + \delta + (1-\alpha) + \alpha + \frac{1}{2} + \delta + \right) \right) \geq \frac{1}{2} + \frac{\delta}{2\nu},$$

where the factor $\frac{1}{3}$ in the above equation appears due to $\texttt{B}$'s choice of playing against one of the three challengers $\texttt{C}_0$, $\texttt{C}_1$ or $\texttt{C}_2$. Now, the fact that the above probability is non-negligible, is a contradiction, which concludes the proof for the case of the cloud.

Now we examine the case of the proxy. Again observe here that the non-collusion assumption allows us to prove Proxy-ORAM's security against the proxy by means of Definiiton 15. Assume thus, that there exists a PPT adversary $\texttt{A}$ that controls the proxy, and can win the IND-CQA game with a non-negligible advantage δ. We will construct a PPT algorithm $\texttt{B}$ that breaks either the semantic security, or the proxy re-encryption security of the encryption scheme. Recall that similar to the cloud's view, the proxy's view consists of the view during the access and the view during a potential Bloom filter creation and reshuffle. All these can be of different lengths, depending on the number of the Proxy-ORAM accesses made. Similar to the case of the cloud, observe that in *every* access the proxy sees at least one encrypted message, and at least one proxy re-encrypted message.

$\texttt{B}$ plays against two challengers: A $\texttt{PREN-IND-CPA}$ challenger $\texttt{C}_0$ and an $\texttt{IND-CPA}$ challenger $\texttt{C}_1$. $\texttt{B}$ also has access to the respective oracles, $\mathcal{O}_0$ and $\mathcal{O}_1$ from which he can ask and receive proxy re-encrypted messages and encrypted messages respectively. $\texttt{B}$ plays the role of a Proxy-ORAM client, and thus creates a plaintext version of the Proxy-ORAM structure (i.e., both the proxy and the cloud) that stores N datablocks. $\texttt{B}$ now encrypts all the structures, by using $\mathcal{O}_1$, and keeps also an encrypted version of all structures. $\texttt{B}$ starts $\texttt{A}$, who asks for the views of the Proxy-ORAM's proxy for data request sequences

he poses. In the pre-challenge phase, each time that B makes a data request sequence, B prepares the corresponding view as follows:

1. Having both $\texttt{BFshare}_0$ and $\texttt{BFshare}_1$, B can find which positions of the Bloom filter he has to retrieve. He uses the encrypted Bloom filter to add these to the view.

2. If a reshuffle is triggered, B runs it in the plaintext view, and simulates it in the encrypted version, using $\mathcal{O}_0$ in order to create the proxy re-encrypted versions when needed.

3. Every time B needs to perform a re-randomisation of an encrypted packet, he uses $\mathcal{O}_1$.

In the challenge phase, A passes to B the tuple $(\texttt{DRS}_0, \texttt{DRS}_1)$. B selects randomly against which of the two challengers he wants to play. B also picks a random bit b^*, makes two copies of the plaintext version of the structures, and runs in the first copy $\texttt{DRS}_0$ and in the second $\texttt{DRS}_1$. If B chose to play against $\texttt{C}_0$, he prepares $\texttt{DRS}_{b^*}$ as in the pre-challenge phase with the following difference: He picks the first position of the plaintext view, where $\texttt{View}(\texttt{DRS}_0)$ and $\texttt{View}(\texttt{DRS}_1)$ differ and need to be proxy re-encrypted, and this way forms the challenge for $\texttt{C}_0$ by sending all the respective encrypted messages $\{c_0^j\}_{j=1}^{\nu}$ and $\{c_1^j\}_{j=1}^{\nu}$. He then picks a random challenge to send to $\texttt{C}_1$. If B chose to play against $\texttt{C}_1$, he prepares $\texttt{DRS}_{b^*}$ as in the previous case, this time finding the messages that need to be re-randomised and sending them to $\texttt{C}_1$. He then picks a random challenge to send to $\texttt{C}_0$. The challengers respond, and B injects the $\texttt{C}_0$'s response in $\texttt{View}(\texttt{DRS}_{b^*})$ if he chose to play against him, discarding the other response. Similarly, B injects $\texttt{C}_1$'s response in the appropriate position of $\texttt{View}(\texttt{DRS}_{b^*})$ if he chose to play against him, and discards the other response. The rest of the $\texttt{View}(\texttt{DRS}_{b^*})$, B creates as in the pre-challenge phase. A now continues with the post-challenge phase, which is ran in a similar way as the pre-challenge. At some point, A returns a bit b' and B then outputs (b', d) if he chose to play against $\texttt{C}_0$, or (d, b') if B chose to play against $\texttt{C}_1$, with d being a random bit.

Observe now, that if $b^* = b$, then A gets a correct view, and can thus win the IND-CQA game with non-negligible advantage δ. Assuming further that each time A gets an incorrect view he outputs 0 with some probability α and 1 with probability $1-\alpha$, we calculate B's total winning probability as

$$\text{Prob}[\texttt{B wins } \texttt{C}_0 \text{ or } \texttt{C}_1] = \frac{1}{2} \cdot \frac{1}{4} \left(2 \cdot \left(\frac{1}{2} + \delta + (1-\alpha) + \alpha + \frac{1}{2} + \delta \right) \right) \geq \frac{1}{2} + \frac{\delta}{2\nu},$$

where again (similar to the cloud's case) the $\frac{1}{2}$ factor occurs due to B's choice to play against $\texttt{C}_0$ or $\texttt{C}_1$. The fact that the above winning probability is non-negligible, is a contradiction, which concludes the proof. □

5.5. A Concrete Instantiation

In order to achieve its design goals, Proxy-ORAM needs to utilise cryptographic building blocks that offer specific properties. The foremost Proxy-ORAM requirement is the use of a semantically secure, multiplicative homomorphic encryption scheme that allows re-randomisation of ciphertexts and proxy re-encryption. One such scheme is the ElGamal encryption scheme [Gam85], which we recall in Section 3.2.1. We will denote the key-pair $(\texttt{skbf}, \texttt{pkbf})$ as the one, under which the responsible share for the creation of the Bloom filter (i.e., c_3) will be encrypted, while $(\texttt{skid}, \texttt{pkid})$ and $(\texttt{sk}, \texttt{pk})$ will be the key-pairs under which the index (i.e., c_2) and the actual datablock is encrypted, respectively. Note that these prerequisites apply mainly for the homomorphic operations that take place regarding the c_1, c_2 and c_3 parts of every datablock packet. For the actual data (i.e., the c_4 part of the packet) the minimum security requirement for the encryption scheme is that it is semantically secure. In fact, as we shall see in Chapter 7, for a use-case like genomic studies, c_4 has to encrypted under an additively homomorphic encryption scheme.

Proposition 7.
Assuming that the DDH is hard for a group G, the ElGamal encryption scheme [Gam85] has computationally indistinguishable ciphertexts under adaptive chosen blinded attack.

Proof. The proof follows from a standard reduction to the DDH. Suppose that the ElGamal cryptosystem, $\mathcal{E}$ is not IND-CBA secure, and thus there exists a PPT adversary $\texttt{A}$, that wins the IND-CBA game from Definition 22 with probability

$$\text{Prob}\left[\mathsf{Game}_{\texttt{A},\mathcal{E}}^{\texttt{IND-CBA}}(\lambda) = 1\right] \geq \frac{1}{2} + \alpha$$

for some non-negligible function α in the security parameter of the ElGamal cryptosystem. Using $\texttt{A}$, we will construct an efficient distinguisher $\texttt{D}$ that can distinguish between tuples of the form (g, g^x, g^y, g^{xy}) and (g, g^x, g^y, g^z), for some random $z \in \langle g \rangle$.

$\texttt{D}$ is handed the tuple $u = \left(\langle p, g\rangle, g^x, g^y, g^z\right)$ for a group generator g of order a safe prime p, and random elements of the group, x, y. $\texttt{B}$ now hands over to $\texttt{A}$ the tuple $\left(\langle p, g\rangle, g^x\right)$ which $\texttt{A}$ can use to encrypt messages of his choice in the first and second learning phases of the game. In each of these phases, $\texttt{A}$ hands over to $\texttt{D}$ an encryption of a random message m which is of the form $c = (g^r, mg^{xr})$, for a random element $r \in \langle g \rangle$. By choosing a random element $t \in \langle g \rangle$, $\texttt{D}$ blinds c and returns to $\texttt{A}$ $c' = (g^{r+t}, mg^{x(r+t)})$.

In the challenge phase, $\texttt{A}$ sends to $\texttt{D}$ the tuple $(c_0, c_1) = ((g^{r_0}, m_0 g^{xr_0}), (g^{r_1}, m_1 g^{xr_1}))$. $\texttt{D}$ chooses a random bit b, and returns to $\texttt{A}$ a blinded version of c'_b, which he computes using g^x and g^z as follows: $c'_b = (g^{r_b} g^y, m_b g^{xr_b} g^z)$. At the end of the game, $\texttt{A}$ replies with a bit b^*, and $\texttt{D}$ wins if $b = b^*$.

Now, if u is a tuple of the form (g, g^x, g^y, g^{xy}), then D wins with the same probability that A wins, thus

$$\text{Prob}[\mathtt{D}(u) = 1] = \text{Prob}\left[\mathsf{Game}^{\text{IND-CBA}}_{\mathtt{A},\mathcal{E}}(\lambda) = 1\right] \geq \frac{1}{2} + \alpha.$$

If on the other hand, u is a random tuple (g, g^x, g^y, g^z), then since we can find a unique $w \in \langle g \rangle$ such that $m_b g^{xr_b} g^z = m_b g^w$ for any element of $\langle g \rangle$, no information about b is given to A, and thus D wins only if b^* which is now randomly chosen, equals b, meaning that

$$\underset{u \leftarrow (g^x, g^y, g^z)}{\text{Prob}}[\mathtt{D}(u) = 1] = \text{Prob}[b^* = b] = \frac{1}{2}.$$

By the above observations, we have that

$$\left| \underset{u \leftarrow (g^x, g^y, g^{xy})}{\text{Prob}}[\mathtt{D}(u) = 1] - \underset{u \leftarrow (g^x, g^y, g^z)}{\text{Prob}}[\mathtt{D}(u) = 1] \right| \geq \alpha,$$

which means that the advantage of D in distinguishing the two types of tuples is non-negligible in the security parameter of $\mathcal{E}$, which contradicts the the DDH assumption. □

Proposition 8.
The ElGamal encryption scheme [Gam85] has computationally indistinguishable proxy re-encrypted ciphertexts under adaptive chosen plaintext attack.

Proof. The proof follows from a standard reduction to the DDH. Suppose that the ElGamal cryptosystem, $\mathcal{E}$ is not PREN-IND-CPA secure, and thus there exists a PPT adversary A, that wins the PREN-IND-CPA game from Definition 25 with probability

$$\text{Prob}\left[\mathsf{Game}^{\text{PREN-IND-CPA}}_{\mathtt{A},\mathcal{E}}(\lambda) = 1\right] \geq \frac{1}{2} + \alpha$$

for some non-negligible function α in the security parameter of the ElGamal cryptosystem. Using A, we will construct an efficient distinguisher D that can distinguish between tuples of the form (g, g^x, g^y, g^{xy}) and (g, g^x, g^y, g^z).

D is handed the tuple $u = \left(\langle p, g \rangle, g^x, g^y, g^z\right)$ for a group generator g of order a safe prime p, and random elements of the group, x, y. B now hands over to A the tuple $\left(\langle p, g \rangle, g^x\right)$ which A can use to encrypt messages of his choice in the first and second learning phases of the game. In each of these phases, A hands over to D an encryption of a random message m which is of the form $c = (g^r, mg^{xr})$, for a random element $t \in \langle g \rangle$. By choosing a random element $s \in \langle g \rangle$, D re-randomises c using g^x getting $c' = (g^{r+t}, mg^{x(r+t)})$, which he further proxy re-encrypts using the key tsk, thus getting $c^* = (g^{r+t}, mg^{x \cdot \mathtt{tsk} \cdot (r+t)})$, which he returns to A.

In the challenge phase, A sends to D the tuple $(c_0, c_1) = ((g^{r_0}, m_0 g^{x r_0}), (g^{r_1}, m_1 g^{x r_1}))$. D chooses a random bit b, and first re-randomises c_b, getting $c'_b = (g^{r_b+y}, m_b g^{x(r_b+y)})$. Then, using tsk, D proxy re-encrypts c'_b, and gets $c^*_b = (g^{r_b+y}, m_b g^{(x+\mathtt{tsk})(r_b+y)})$, which he sends to A.

Now, if u is a tuple of the form (g, g^x, g^y, g^{xy}), then D wins with the same probability that A wins, thus

$$\text{Prob}[\mathtt{D}(u) = 1] = \text{Prob}\left[\mathsf{Game}^{\mathtt{PREN\text{-}IND\text{-}CPA}}_{\mathtt{A},\mathcal{E}}(\lambda) = 1\right] \geq \frac{1}{2} + \alpha.$$

If however, u a random tuple (g, g^x, g^y, g^z), then since we can find a unique $w \in \langle g \rangle$ such that $m_b g^{x r_b} g^z = m_b g^w$ for any element of $\langle g \rangle$, no information about b is given to A, and thus D wins only if b^* which is now randomly chosen, equals b, meaning that

$$\underset{u \leftarrow (g^x, g^y, g^z)}{\text{Prob}}[\mathtt{D}(u) = 1] = \text{Prob}[b^* = b] = \frac{1}{2}.$$

By the above observations, we have that

$$\left| \underset{u \leftarrow (g^x, g^y, g^{xy})}{\text{Prob}}[\mathtt{D}(u) = 1] - \underset{u \leftarrow (g^x, g^y, g^z)}{\text{Prob}}[\mathtt{D}(u) = 1] \right| \geq \alpha,$$

which means that the advantage of D in distinguishing the two types of tuples is non-negligible in the security parameter of $\mathcal{E}$, which contradicts the the DDH assumption. □

Furthermore, Proxy-ORAM needs to utilise a homomorphic, cryptographically secure hash function. One such hash function is the Chaum-van Heisjt-Pfitzmann [CHP92] hash function, as we have seen in Chapter 3.2.1. We instantiate the CvHP by selecting two group generators g_1, g_2 for the group $\mathbb{Z}^*_p$ for a cryptographically secure prime p. For a datablock with identifier id, we set the first index share to be $\mathtt{IDshare}_0 = g_1^{\mathcal{H}(l \| r(l) \| c)}$, where l is the level to which the datablock is stored, $r(l)$ is a counter of the reshuffles done at level l thus far, and c is a counter, initiated to a random value chosen by the client and subsequently incremented every time a reshuffle is performed at any level of Proxy-ORAM. The second index share is formed as $\mathtt{IDshare}_1 = g_2^{\mathsf{id}}$ and thus the index of datablock with identifier id at level l, after $r(l)$ reshuffles at level l, and after c reshuffles have taken place in Proxy-ORAM in total is

$$\mathtt{index}(\mathsf{id}, r, r(l), c) = g_1^{\mathcal{H}(l \| r(l) \| c)} g_2^{\mathsf{id}}.$$

For creating the Bloom filter we use again the CvHP hash function, this time on two group generators g_3 and g_4 of the group $\mathbb{Z}^*_p$ for a safe prime p. For the ξ keys needed for the Bloom filter, we choose random

	$g_1^{\mathcal{H}}$	g_2^{id}	$g_3^{k_i}$	g_4^{id}	`skid`	`sktemp`	`skbf'`
Client	•	•	•	•	•	•	•
Investigator	•	•		•			
Cloud							•
Proxy	•		•		•	•	

Table 5.2: Overview of the values known to every party, for a specific datablock, level, and after a specific number of reshuffles. $g_1^{\mathcal{H}(l\|r(l)\|c)}$, g_2^{id}: Shares for computing the current `index` of the datablock, $g_3^{k_i}$, g_4^{id}: Shares for determining the Bloom filter positions for the datablock; `skid`: secret key to decrypt a packet's encrypted c_2 part; `skbf`: secret key to decrypt a packet's encrypted c_3 part; `skbf'`: secret key to decrypt encrypted Bloom filter bits;

values from the group – recall that each level has its own set of ξ keys, renewed in every reshuffle and stored on the proxy. We thus form the positions of the Bloom filter that should be set to '1' as

$$\mathtt{CvHP}(k_i, \mathsf{id}) = g_3^{k_i} g_4^{\mathsf{id}}, \text{ for } i \in \{1, \ldots, \xi\}.$$

Note that the output of the `CvHP` hash function is of size $|p|$ while the Bloom filter will be much smaller. Suppose that $b(l)$ is the size of the Bloom filter at level i as indicated by the security parameter λ. One way of performing the hash is by taking the first $|b(l)|$ bits of the hash function's output. Hence, without loss of generality, we will abuse notation, and from now on, by $\mathtt{CvHP}(k_i, \mathsf{id})$ we will mean $\mathtt{CvHP}(k_i, \mathsf{id})$ && (($1 \ll |b(l)|$) -1).

Recalling that the two shares (one 'responsible' for the Bloom filter creation and the other for the index creation), as well as the data have to be encrypted, the packet (cf. Definition 21) for a datablock with identifier id is the following tuple: $\mathsf{packet}(\mathtt{index}(\mathsf{id})) = (c_1, c_2, c_3, c_4)$, with

$$\left(g_1^{\mathcal{H}(r(l)\|l\|c)} g_2^{\mathsf{id}}, \quad \mathtt{Enc}_{\mathtt{pkid}}(g_2^{\mathsf{id}}), \quad \mathtt{Enc}_{\mathtt{pkbf}}(g_4^{\mathsf{id}}), \quad \mathtt{Enc}_{\mathtt{pk}}(\mathtt{dat}) \right).$$

In Section 5.2, we have seen an abstract exposition of the protocols used in Proxy-ORAM. Here, we describe only the points of each protocol that need to be adjusted for the particular instantiation.

During initialisation, the client generates the encryption scheme key pairs and the parameters for the CvHP hash function. The client then distributes the shares and the secret keys as described in Section 5.2.1 and summarised in Table 5.2.

In Step 7 of the Access Protocol (Protocol 3), the investigator sends a blinded version of his share, g_4^{id}. This is done by forming $g_4^{\mathsf{id}+\nu}$ for a randomly chosen value ν. The proxy, knowing the Bloom filter keys $\{k_i\}_{i=1}^{\xi}$ can thus form the values $\{g_3^{k_i} g_4^{\mathsf{id}+\nu}\}_{i=1}^{\xi}$, which he sends back to the investigator, along with $\mathcal{H}(r(l)||l||c)$. The investigator can now remove the blindings from the values he received and thus find

the positions of the Bloom filter that he has to check. The contents of these Bloom filter positions are sent from the proxy to the investigator. In turn, the investigator blinds the received values (each with a different blinding value) and sends them to the cloud who decrypts them, since he is in possession of $\mathtt{skbf}$. The cloud then encrypts these values using the public key $\mathtt{pktemp}$ and sends them back to the investigator. The investigator, having the authentication token $\mathtt{sktemp}$, decrypts the received values, removes the blindings, and thus can learn if the datablock lies on that particular level. If the datablock has already been found on a previous level, or if the datablock is not on the level currently being examined, then the investigator computes the index of the fake packet and requests from the cloud the $\mathsf{packet}(\mathtt{index}(\mathsf{id}_{\text{fake}}))$. Otherwise, the investigator computes the index of the desired datablock and requests from the cloud the $\mathsf{packet}(\mathtt{index}(\mathsf{id}))$. The element c_2 of the packet requested from the investigator, the cloud proxy re-encrypts, and sends the whole packet to the investigator.

In the Bloom filter creation phase, the cloud uses proxy re-encryption in order to transform the c_3 element of every packet (step 14 of Protocol 4). For the key pair $(\mathtt{tsk}, \mathtt{tpk})$, generated by the cloud, this is done as follows: the 'new' public key is set to be $\mathtt{pkbf} \cdot \mathtt{tpk}$, and $\mathtt{tsk}$ is set as the re-encryption key, so that $\mathtt{skbf} + \mathtt{tsk}$ is the new decryption key. Thus, c_3 is set to be $\mathtt{Enc}_{\mathtt{pk}\cdot\mathtt{tpk}}(g_4^{\mathsf{id}})$. Now the proxy can compute $\mathtt{Enc}_{\mathtt{pk}\cdot\mathtt{tpk}}(g_3^{k_i})$ and thus form $\mathtt{Enc}_{\mathtt{pk}\cdot\mathtt{tpk}}(g_3^{k_i} g_4^{\mathsf{id}})$, to which the proxy removes the $\mathtt{skbf}$ dependency by computing $\mathtt{Dec}_{\mathtt{sk}}(\mathtt{Enc}_{\mathtt{pk}\cdot\mathtt{tpk}}(g_3^{k_i} g_4^{\mathsf{id}}))$, and sending these values to the cloud. The cloud can now further decrypt using his $\mathtt{tsk}$

$$\mathtt{Dec}_{\mathtt{tsk}}(\mathtt{Enc}_{\mathtt{tpk}}(g_3^{k_i} g_4^{\mathsf{id}})) = g_3^{k_i} g_4^{\mathsf{id}},$$

which is a position in the Bloom filter that has to be set to '1'.

In the update phase, the cloud generates a new temporary ElGamal key pair $(\mathtt{tsk}, \mathtt{tpk})$ and having removed the c_1 part of every packet to be held by $\mathtt{level}_l$, he proxy re-encrypts the packet's remaining parts in the same way he did in the Bloom filter creation phase. The cloud sends the proxy re-encrypted parts, as well as $\mathtt{tpk}$ to the proxy. The latter increases the counters c and $r(l)$ and computes $\mathcal{H}(r(l)||l||c)$. For each received element, the proxy then multiplies the packet's c_2 part $\mathtt{Enc}_{\mathtt{pkid}\cdot\mathtt{tpk}}(g_1^{\mathcal{H}(r(l)||l||c)})$, which results in the new correct index c_1. This index however is now encrypted under $\mathtt{pkid} \cdot \mathtt{tpk}$. The proxy then removes the dependency on the $\mathtt{pk}$ from these encrypted indices by using $\mathtt{skid}$ as the re-encryption key (similarly to what was done in the Bloom filter creation phase).

Extra care has to be taken also regarding the size of the Bloom filter used for an instantiation of Proxy-ORAM. The first estimation for the false positive ratio was given by Bloom as

$$p = \left(1 - \left(1 - \frac{k}{m}\right)^n\right)^k$$

for a Bloom filter of size m, storing n items and using k hash functions. Recently however, it has been shown by Bose et al. [Bos+08] that this original rate estimation was false. Christensen et al.[CRJ10] showed that the false positive is

$$p = \sum_{i=1}^{m} \left(P(kn, m, i) \left(\frac{i}{m} \right)^k \right),$$

where $P(N, M, K)$ is the probability of having exactly K Bloom filter positions set to '0' when N items are stored in a Bloom filter of size M. In order to construct a Bloom filter that has a false positive ratio negligible in the security parameter of an encryption scheme (i.e., close to 2^{-120}), one would have to set the Bloom filter size polynomially larger than the amount of the stored datablocks. However, having experimentally observed that Bloom filters of linear size still yield false positives with far smaller probability than the one theoretically expected, we believe that for practical applications one can tolerate smaller false positive ratios than the ones predicted, as was the case with the privacy preserving filesystem of Williams et al. [WST12].

We close this chapter with a discussion on the creation of the last level's Bloom filter, which will play an important role in contruction's practical evaluation.

The purpose of the Bloom filter is to inform the investigator whether a given packet is to be found on a specific level or not. Since in the initialisation of Proxy-ORAM all packets are originally uploaded to the last level, we know that any packet can surely be found on the last level before it is reshuffled for the first time (which happens only after N queries). So as long as we perform less than N queries, a Bloom filter for the last level is not needed: Every element (potentially older versions) will be found on the last level by construction. In other words a Bloom filter would return only 1s, which makes it redundant. Thus, in the original upload we can avoid creating the last level's Bloom filter.

If we perform more than N queries, a reshuffle of the last level will automatically be triggered, which will create a Bloom Filter for that level and make it the second last level (again, the last level then does not need a Bloom Filter).

Note that this optimisation does not affect the ORAM's operations as described in the previous sections: if a packet has not been found on any of the higher levels, then this packet has never been retrieved in the ORAM's history and therefore it *will* be found in the last level. The same holds for the case of a `Write` operation. The `Reshuffle` operation is also not affected, since before a `Reshuffle` begins, the Bloom filters of both levels to be reshuffled are destroyed.

Chapter 6

From one client to many: The case of Blurry-ORAM

Our second construction addresses problems of ORAM architectures that allow multiple clients to store their data on the same server. Here, we identify, analyse and rigorously treat this problem, having the following question as our starting point:

Can we build an ORAM that allows multiple clients to store their private data on a remote server, while allowing each of them to partially share any part of his data with any other client?

We answer the above question in an affirmative way by presenting the first ORAM construction that achieves this goal with low communication complexity between the involved clients and the server under the assumption that all involved parties are semi-honest and not colluding. At the same time, we are the first to rigorously define privacy between clients in such a setting. Indeed, as we shall see, ORAMs in multi-client environments have been developed in the past. However, all but one of the previous constructions allow only full (and not partial) sharing of data, while none of the existing solutions proposes a concrete security model that can capture the information leakage potentially occurring between the clients who share data. We note the importance of partial data sharing, as we believe that it is a very realistic setting that can have applications extending from filesystem sharing and private interprocess communication to genomic studies at mass.

Published Content

All the results from this chapter first appeared in [KPK17], which is a joint work with Andreas Peter and Stefan Katzenbeisser. The development of the architecture was jointly done between myself, Andreas Peter and Stefan Katzenbeisser, and the security proofs were done by myself.

6.1. Multi-client ORAMs

The problem of constructing privacy preserving multi-client storage solutions has been explored in a number of works [ZZQ16; Fra+12; Goo+12b; Maf+15; BMN17; Bac+16; WST12; Lor+12; BCP16]. However, as we have already seen in Chapter 2, not all constructions deal with the same problem. The most common and very well studied setting for multi-client ORAMs is the one dealt with in [Maf+15; BMN17; Bac+16]; there, multiple clients share the same dataset and want to access it in a way that protects the client access patterns against the server, while at the same time the client's identity is hidden from the server. We will refer to this setting as 'access pattern anonymity'. The second multi-client ORAM setting is the one coined as 'Parallel ORAM', which has been extensively studied by Boyle et al. [BCP16] and Lorch et al. [Lor+12]. In that setting, again one dataset (fully owned by every client) is accessed obliviously by multiple clients, however this time (and as the name implies) in *parallel*. The above results delivered theoretical constructions and bounds, while Williams et al. [WST12] proposed a practical parallel ORAM construction that resulted in the first access pattern hiding filesystem.

The problem we are dealing with however is closer to the setting proposed by Franz et al. [Fra+12]. There, a client wants to give *partial access* of his data to other clients, making sure that the client access patterns are hidden from the server. In this chapter, we take the setting from [Fra+12] one step further: We consider one server and multiple clients, each of which shares only parts of his data with other clients of his choice, while the access patterns of every client remain hidden from the server, and *all other clients*. The basic architectural structure is depicted in Figure 6.2. We will call an ORAM that achieves this property a 'Partial Sharing Multi-Client ORAMs (PSMC-ORAMs)'. Both the server and the clients are assumed to be semi-honest and non-colluding with each other, i.e., they try to extract as much information possible about the data belonging to other clients, but they never deviate from the protocol.

Using our new ORAM framework from Chapter 4, we define the PSMC-ORAM, and then construct one such ORAM architecture, named Blurry-ORAM. Our construction is based on the highly efficient Path-ORAM construction of Stefanov et al. [Ste+13a], and thus inherits from the latter its simplicity and efficiency. In Table 6.1 we sum up the properties of existing multi-client ORAM constructions, show their communication complexity and recall if they allow access pattern privacy against other clients, allow for anonymous accesses or sharing of data between the clients. To our knowledge, our construction is the first to allow partial data sharing between clients in an ORAM, and at the same time guarantee access pattern hiding not only against the server, but also against other clients.

6.2. Partial Sharing Multi-Client-ORAMs

We begin by forming the necessary theoretical basis with which we can describe multi-client ORAM constructions. For this, we will make use of our new framework from Chapter 4, which we will adjust to

Solution	Commun.Complexity	A.P. Hiding (Server)	A.P. Hiding (Clients)	A.P. Anonymity	Data sharing
[Fra+12]	$O(\sqrt{N}\log^2 N)$ (amort.)	✓	×	×	✓
[Goo+12b]	$O(\log^2 N)$	✓	×	×	✓
[ZZQ16]	$O(\log N \log\log N)$ (amort.)	✓	×	×	×
[BMN17]	$O(\log^2 N)$ to $O(\log^5 N)$	✓	×	×	✓
[Maf+15]	$O(G\log^2 N)$	✓	×	✓	✓
[Bac+16]	$O(\log^2(KN))$	✓	×	✓	×
Blurry-ORAM	$O(K\log^2 N\log(\log N))$	✓	✓	×	✓

Table 6.1: Comparison of multi-client ORAM solutions; N: Number of blocks per client, G: Number of groups, K: Number of clients.

the multi-client setting. Observe here once again that it is the flexibility of our new framework that allows us to define in a rigorous and yet practical way a more complex system such as a multi-client ORAM.

Similar to Chapter 4, we consider here all parties (i.e., the server and every client) as PPT Turing machines. All the clients communicate with the server through a tape Ξ, and every two clients share a communication tape ξ. On a high level, a multi-client ORAM that allows partial data sharing must provide protocols that allow the initialisation of the architecture, reading and writing data on the server, as well as protocols that allow sharing data between clients, and revoking formerly given sharing rights.

Definition 26 (Multi-Client Data Request)**.**
A multi-client data request $\mathtt{mcdr}$ *is a tuple of the form* $(\mathsf{op}, \mathsf{id}, \mathtt{dat}, \mathtt{Client}_j)$ *where* $\mathsf{op} \in \{\mathtt{Read}, \mathtt{Write}, \mathtt{Share}, \mathtt{Revoke}\}$, id *is the datablock's identifier,* $\mathtt{dat}$ *is the data to be written, and* $\mathtt{Client}_j$ *is the client with which* $\mathsf{block}_{\mathsf{id}}$ *will be shared with, or from which sharing of* $\mathsf{block}_{\mathsf{id}}$ *will be revoked. If* $\mathsf{op} = \mathtt{Read}$*, then* $\mathtt{dat} = \perp$ *and* $\mathtt{Client}_j = \perp$*. If* $\mathsf{op} = \mathtt{Write}$*, then* $\mathtt{Client}_j = \perp$*. If* $\mathsf{op} \in \{\mathtt{Share}, \mathtt{Revoke}\}$*, then* $\mathtt{dat} = \perp$ *and* $\mathtt{Client}_j$ *is the client with which the block will be shared, or from whom the sharing will be revoked.*

Definition 27.
A multi-client data request sequence is a tuple of the form $(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_l)$*, where each* $\vec{x}_i$ *is a multi-client data request. The number* l *of data requests in a multi-client data request sequence is called the data request sequence's length.*

Definition 28 (Server communication transcript)**.**
The server's communication transcript com_t^S *at round* t *of the protocol is the content of the communication channel* Ξ *at round* t *of the protocol's execution.*

Definition 29 (Client communication transcript)**.**
The client's communication transcript com_t^C *at round* t *of the protocol is the content of the communication channel* Ξ *concatenated with the contents of all* $\{\Xi_i\}_{i=1}^{\tilde{K}}$ *at round* t *of the protocol's execution, where* Ξ_i*,* $i = 1, \ldots, \tilde{K}$*, is the content of the client's communication tape with another client.*

In the above definition, observe that since the number of clients is finite, the same holds for the number of communication tapes a client has, $\tilde{K}$.

Similar to Definition 30, we define here the view of the client by means of a partial ordering imposed on the communication transcript.

Definition 30 (View of the Server and view of the Client)**.**
Let $\leq$ denote a partial order over the set of communication transcripts $A = \{\mathsf{com}_1, \mathsf{com}_2, \ldots, \mathsf{com}_T\}$ for an ORAM protocol of T rounds such that for any $\mathsf{com}_i \in A$, $\mathsf{com}_j \in A$, $\mathsf{com}_z \in A$ the following hold:

1. *For any $\mathsf{com}_i \in A$, $\mathsf{com}_i \leq \mathsf{com}_i$.*
2. *For any $\mathsf{com}_i \in A$, $\mathsf{com}_j \in A$, if $\mathsf{com}_i \leq \mathsf{com}_j$ and $\mathsf{com}_j \leq \mathsf{com}_i$, then $\mathsf{com}_i = \mathsf{com}_j$.*
3. *For any $\mathsf{com}_i \in A$, $\mathsf{com}_j \in A$, $\mathsf{com}_z \in A$, it holds that if $\mathsf{com}_i \leq \mathsf{com}_j$ and $\mathsf{com}_j \leq \mathsf{com}_z$, then $\mathsf{com}_i \leq \mathsf{com}_z$.*

For a PSMC-ORAM protocol of T rounds, the server's view is the union of all server communication transcripts from round 1 to T: $\mathtt{View}^S = \cup_{t=1}^{T} \mathsf{com}_t^S$, and the client's view is the union of all client communication transcripts $\mathtt{View}^C = \cup_{t=1}^{T} \mathsf{com}_t^C$

Definition 31 (Shared Block)**.**
In a multi-client ORAM construction that allows data sharing between clients, we call a block id_i shared if at least two clients have access to it.

Definition 32 (Partial Sharing Multi-Client ORAM)**.**
Let $\tilde{N} \in \mathbb{N}$, $K \in \mathbb{N}$, $M \geq D$ and $\mathcal{E} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$ be a secret-key encryption scheme mapping M-bit plaintexts to B-bit ciphertexts. A Partial Sharing Multi-Client ORAM (PSMC-ORAM) $\mathtt{ORAM}_{\mathcal{E}}$ with parameters $(D, \tilde{N}, K, \mathcal{E})$ is a set of multi-party interactive randomised algorithms, $(\mathtt{PSMC\text{-}ORAM.Init}, \mathtt{PSMC\text{-}ORAM.Access})$, run by a server S and clients $\mathtt{Client}_1, \ldots, \mathtt{Client}_K$, such that:

- $\mathtt{PSMC\text{-}ORAM.Init}(\lambda, N, K) \rightarrow (\mathtt{Client}_1, \cdots, \mathtt{Client}_K, \mathtt{S})$ *in the following way:*
  1. *λ is the security parameter, K is the number of clients, every client has N datablocks and stores on the server $\tilde{N} > N$ datablocks;*
  2. *$\mathtt{key} \leftarrow \mathtt{KeyGen}(1^\lambda)$ is generated by every client $\mathtt{Client}_i$ for $i = 1$ to K;*
  3. *S includes a database $S.DB = (\mathsf{block}_1, \ldots, \mathsf{block}_{K\tilde{N}})$.*
- $\mathtt{PSMC\text{-}ORAM.Access}(\mathtt{Client}_1, \ldots, \mathtt{Client}_K, \mathtt{S}, \mathtt{mcdr}) \rightarrow (\mathtt{Client}'_1, \cdots, \mathtt{Client}'_K, \mathtt{S}', \mathsf{com})$ *in the following way:*
  1. *For some $i = 1, \ldots, K$, $\mathtt{Client}_i$ issues a data request $\mathtt{mcdr}$;*
  2. *$\mathtt{Client}_i$ and S communicate through Ξ, producing the communication's transcript com.*

Similar to Definition 32 from Chapter 4, we need to address here the soundness of a PSMC-ORAM. We do this in the next definition, again avoiding going into much details, but rather describing the minimal properties, so that the ORAM 'works'.

Definition 33 (Minimal Partial Sharing Multi-Client-ORAM Soundness Conditions)**.**
A PSMC-ORAM construction $\texttt{ORAM}_{\mathcal{E}}$ *has minimal soundness if the following hold:*

1. *For any* (λ, N, K)*, if* $(\texttt{Client}_1, \cdots, \texttt{Client}_K, \texttt{S}) \leftarrow \texttt{PSMC-ORAM.Init}(\lambda, N, K)$*, then every client's* $\texttt{Client}_i$ *secret key* $\texttt{key}_i$ *from Definition 32 must be accessible by* $\texttt{Client}_i$*.*
2. *For any* $\texttt{mcdr} = (\mathsf{op}, i, \mathsf{data}, \texttt{Client}_j)$*, if* $(\texttt{Client}'_1, \cdots, \texttt{Client}'_K, \texttt{S}', \mathsf{com}) \leftarrow \texttt{PSMC-ORAM.Access}(\texttt{Client}_1, \cdots, \texttt{Client}_K, \texttt{mcdr})$*, then:*
   a) *for every client/secret key pair,* $(\texttt{Client}_i, \texttt{key}_i)$ *with* $i = 1, \ldots, K$*, it holds that if the secret key* $\texttt{key}_i$ *of* $\texttt{Client}_i$ *is accessible by* $\texttt{Client}_i$*, then it is also accessible by* $\texttt{Client}'_i$*;*
   b) *if* $\mathsf{op} = \texttt{Read}$ *and* $S.DB(u) = \mathsf{block}$*, then* $\mathsf{Data}(\mathsf{block})$ *must be accessible by* $\texttt{Client}'_i$*;*
   c) *if* $\mathsf{op} = \texttt{Write}$ *and* $S'.DB(u) = \mathsf{block}$*, then* $\mathsf{Data}(\mathsf{block}) = \mathsf{data}$*.*
   d) *if* $\mathsf{op} = \texttt{Share}$*, and* $S.DB(u) = \mathsf{block}$ *is accessible by* $\texttt{Client}_i$ *but not from* $\texttt{Client}_j$*, then it must be accessible from both* $\texttt{Client}'_i$ *and* $\texttt{Client}'_j$*;*
   e) *if* $\mathsf{op} = \texttt{Revoke}$ *and* $S.DB(u) = \mathsf{block}$ *is accessible by* $\texttt{Client}_i$ *and from* $\texttt{Client}_j$*, then it must be accessible from* $\texttt{Client}'_i$ *but not from* $\texttt{Client}'_j$*.*

6.3. Security Definition in a Multi-Client Environment

In typical ORAM constructions, the model considered includes simply a client and a server, yet as we have already seen, hiding the access patterns against the server is already a non-trivial problem. Thus, it does not come as a surprise that when multiple clients are present and share parts of their data with each other, the situation becomes much more involved. In this section, we analyse how a semi-honest client can behave, and shed some light to the problem of client-to-client privacy.

We consider a semi-honest server and semi-honest clients that do not collude with each other or with the server. Thus, all involved parties do not deviate from the protocol but try to gain as much information as possible (e.g., which data blocks were read or written by which client, how many blocks are shared with whom, etc.) by examining the views of the access transactions. In order to model this behaviour, we first have to adjust Definition 14 to the new setting of the PSMC-ORAM.

Definition 34 (Access Pattern for Partial Sharing Multi-Client-ORAMs)**.**
Given a PSMC-ORAM client $\texttt{C}$*, a server* $\texttt{S}$*, and a multi-client data request* $\texttt{mcdr}$ *according to Definition 26, the access pattern* $\texttt{ap}(\texttt{mcdr})$ *is the tuple* $\left(S.DB, \mathsf{com}, S'.DB\right)$*, where*

$$(\texttt{Client}'_1, \cdots, \texttt{Client}'_K, \texttt{S}', \mathsf{com}) \leftarrow \texttt{PSMC-ORAM.Access}(\texttt{Client}_1, \cdots, \texttt{Client}_K, \texttt{S}, \texttt{mcdr}).$$

6.3.1. Security Against the Server

We begin our analysis by first considering security against the server. We do this by defining a variant of the IND-CQA game from Definition 15. The main difference comes from the fact that in Definition 15, the adversary (the server) was playing against one client who could only read or write datablocks on the server's database. Here however the scenario is different: The adversary is playing against a multitude of clients who store and share datablocks with each other. Thus in any point of the game (including the challenge phase), the adversary should be allowed to ask *any* client to perform an operation, such as read or write one of his datablocks, or share datablocks with other clients. Taking these observations into account, we define the PSMC-ORAM server/client game as follows.

Definition 35 ($\mathsf{Game}^{\texttt{Server-IND-CQA}}_{\mathsf{A},\mathsf{ORAM}}(\lambda)$)**.**
Let $\texttt{PSMC-ORAM} = (\texttt{PSMC-ORAM.Init}, \texttt{PSMC-ORAM.Access})$ *be a Partial Sharing Multi-Client ORAM construction,* λ *a security parameter and* A *an ORAM adversary. The computational indistinguishability of access patterns game under adaptive chosen query attack* $\mathsf{Game}^{\texttt{Server-IND-CQA}}_{\mathsf{A},\mathsf{ORAM}}(\lambda)$ *is played between* A *and a simulator* C*, who can execute any data request on behalf of any client that stores datablocks on the* PSMC-ORAM *and proceeds as follows.*

1. A *chooses* $N \leq \tilde{N}$ *and* $K \in \mathbb{N}$*;*
2. $(\texttt{Client}_1, \cdots, \texttt{Client}_K, \texttt{S}) \leftarrow \texttt{ORAM.Init}(\lambda, N, K)$*;*
3. *First CQA learning phase: for* $i = 1, \ldots, q_1 \in \mathbb{N}$*,* A *repeats (adaptively) the following:*
   a) A *chooses a data request* $\texttt{mcdr}_i$ *for some client of his choice,* $\texttt{Client}_u$*;*
   b) C *simulates* $\texttt{Client}_u$ *and executes* ORAM.Access *on* $\texttt{mcdr}_i$*;*
   c) A *receives* $\texttt{ap}(\texttt{mcdr}_i)$*;*
4. *Challenge phase:* A *chooses two data requests:* $\texttt{mcdr}^0$ *for* $\texttt{Client}_u$ *and* $\texttt{mcdr}^1$ *for* $\texttt{Client}_w$*;*
5. C *chooses randomly a secret bit* $b \xleftarrow{\$} \{0,1\}$ *and by simulating the respective client, he executes* ORAM.Access *on* $\texttt{mcdr}^b$*;*
6. A *receives* $\texttt{ap}(\texttt{mcdr}^b)$*;*
7. *Second CQA learning phase: for* $j = 1, \ldots, q_2 \in \mathbb{N}$*,* A *repeats (adaptively) the following:*
   a) A *chooses a data request* $\texttt{mcdr}_j$ *for some client of his choice,* $\texttt{Client}_u$*;*
   b) C *simulates* $\texttt{Client}_u$ *and executes* ORAM.Access *on* $\texttt{mcdr}_j$*;*
   c) A *receives* $\texttt{ap}(\texttt{mcdr}_j)$*;*
8. A *outputs a bit* b'*.*

A *wins the game if and only if* $b = b'$*.*

Definition 36 (Client-to-Server Privacy)**.**
We say that a PSMC-ORAM protocol provides Client-to-Server privacy (i.e., hides the access patterns of a client against the server), if for every PPT PSMC-ORAM adversary $\mathtt{A}$*:*

$$\left|\text{Prob}\left[\mathsf{Game}^{\textit{Server-IND-CQA}}_{\mathtt{A},\mathtt{ORAM}}(\lambda)=1\right]-\frac{1}{2}\right|\leq \textit{negl}(\lambda).$$

Observe that in the challenge phase, no restriction is imposed upon the choice of the client. The adversary can choose to form his challenge on data requests performed by different clients, or by the same client.

6.3.2. Security Against Other Clients

When examining the client's security against the server the transition from the one client to the multiple client model seemed to be rather trivial. In fact, we simply had to adjust the game-based security Definition 15 to the Partial Sharing Multi-Client ORAM in order to form the new security definition. Most importantly, we allowed the adversary to be able to ask for *any* operation (including share and revoke operations) at the challenge phase. The intuition about the scheme's security in such a setting comes from the fact that the server sees only encrypted datablocks during each access, and thus, as long as the length of every view remains the same, the security of the scheme can easily fall back to that of the encryption scheme.

This however is not the case when we examine the security against other clients. In a PSMC-ORAM, the view of a client includes also unencrypted datablocks. At first, these plain datablocks are only the client's datablocks. However, as the client is allowed to share datablocks with and from other clients, the amount of plain datablocks visible by the client changes over time, since the client sees in plain not only his datablocks, but also datablocks that belong to other clients. Now recall that (as we have seen until now in more occasions) during every ORAM access only a small amount of datablocks are moved in the server's database – in the case of Path-ORAM for example, only the $\log N$ datablocks found in the currently accessed path are (potentially) moved. In fact, even from those few blocks in the path, only some will change position within the path, while most of them will be stagnant. This behaviour (inherent in any ORAM construction we know) can be used in a multi-client setting by an adversary to his advantage in attempting to break the access pattern hiding of other clients.

As a consequence of the above, suppose that we would (wrongly) define client-to-client privacy in an (almost) identical way as we did in Definition 35, by means of the following game, played between an adversarial client $\mathtt{A}$, and a simulator $\mathtt{C}$ who can execute any data requests for any client except $\mathtt{A}$. We call this game the $\mathsf{Game}^{\mathtt{Client\text{-}IND\text{-}CQA\text{-}Wrong}}_{\mathtt{A},\mathtt{ORAM}}(\lambda)$, and it would proceed as follows:

1. $\mathtt{A}$ chooses $N \leq \tilde{N}$ and $K \in \mathbb{N}$;
2. $(\mathtt{Client}_1, \cdots, \mathtt{Client}_K, \mathtt{S}) \leftarrow \mathtt{ORAM.Init}(\lambda, (K-1)N)$, except for some client $\mathtt{Client}_\nu$ who is the adversary;

3. First CQA learning phase: for $i = 1, \ldots, q_1 \in \mathbb{N}$, $\mathtt{A}$ repeats (adaptively) the following:
   a) $\mathtt{A}$ chooses a data request $\mathtt{mcdr}_i$ for some client of his choice, $\mathtt{Client}_u$, or for himself;
   b) If the data request was for another client, then $\mathtt{C}$ simulates $\mathtt{Client}_u$ and executes $\mathtt{ORAM.Access}$ on $\mathtt{mcdr}_i$; Otherwise, $\mathtt{A}$ executes the data request himself;
   c) $\mathtt{A}$ receives $\mathtt{ap}(\mathtt{mcdr}_i)$;
4. Challenge phase: $\mathtt{A}$ chooses two data requests: $\mathtt{mcdr}^0$ for $\mathtt{Client}_u$ and $\mathtt{mcdr}^1$ for $\mathtt{Client}_w$, with $u \neq v$ and $w \neq v$;[1]
5. $\mathtt{C}$ chooses randomly a secret bit $b \xleftarrow{\$} \{0, 1\}$ and by simulating the respective client, he executes $\mathtt{ORAM.Access}$ on $\mathtt{mcdr}^b$;
6. $\mathtt{A}$ receives $\mathtt{ap}(\mathtt{mcdr}^b)$;
7. Second CQA learning phase: for $j = 1, \ldots, q_2 \in \mathbb{N}$, $\mathtt{A}$ repeats (adaptively) the following:
   a) $\mathtt{A}$ chooses a data request $\mathtt{mcdr}_j$ for some client of his choice, $\mathtt{Client}_u$, or for himself;
   b) If the data request was for another client, then $\mathtt{C}$ simulates $\mathtt{Client}_u$ and executes $\mathtt{ORAM.Access}$ on $\mathtt{mcdr}_i$; Otherwise, $\mathtt{A}$ executes the data request himself;
   c) $\mathtt{A}$ receives $\mathtt{ap}(\mathtt{mcdr}_j)$;
8. $\mathtt{A}$ outputs a bit b'.

$\mathtt{A}$ wins the game if and only if $b = b'$.

Observe that an adversary $\mathtt{A}$ playing the above game could trivially win it by using the $\mathtt{Share}$ and $\mathtt{Revoke}$ operations in a variety of ways.

One approach would be for $\mathtt{A}$ to $\mathtt{Share}$ a datablock from another client in the pre-challenge phase, and form his challenge by asking a $\mathtt{Read}$ operation on that previously shared datablock and on another datablock which is not his and he does not share. Clearly in such a case, $\mathtt{A}$ would trivially win the game, since if the challenger had performed the $\mathtt{mcdr}$ with the shared datablock, $\mathtt{A}$ would see this particular datablock in the view.

In a similar way, $\mathtt{A}$ could form his challenge by a $\mathtt{Share}$ operation on a block and a $\mathtt{Read}$ on another, and then in the post-challenge phase simply ask for a $\mathtt{Read}$ on the datablock he wanted to share in the challenge phase. If $\mathtt{A}$ could see this datablock in the view plain, then he would know that the challenger chose the data request with the $\mathtt{Share}$ operation.

Yet another way for $\mathtt{A}$ to trivially win the above game would be to form his challenge, and at the post-challenge phase to ask for $\mathtt{Share}$on one of the datablocks of his challenge. The reason why this attack would be successful is the following: Suppose that the PSMC-ORAM were based on the hierarchical solution, and in the challenge phase $\mathtt{A}$ asked for a $(\mathtt{Read}, i, \perp, \perp)$ or a $(\mathtt{Read}, j, \perp, \perp)$. Once one of

[1] Observe that this is a minimum restriction that we have to impose, as otherwise $\mathtt{A}$ who *is* $\mathtt{Client}_v$ would trivially win the game.

these data requests is performed, either datablock i or j would be put on the top level of the hierarchical structure. Now $\mathtt{A}$ asks for $(\mathtt{Share}, i, \perp, \mathtt{A})$, and then performs one more $(\mathtt{Read}, i, \perp, \perp)$. If now $\mathtt{A}$ sees datablock i on the top level, he will immediately know which data request was performed by the challenger.

Observe that what abstractly binds together the above attacks is the fact that according to our model, clients that share a datablock have full control over it. Furthermore, in order to address the two last attacks, involving the information leaked during the second learning phase of the game, one would have to make the overall scheme forward secure. We believe that this could be achieved by employing ideas from identity based encryption, however our primary goal has been to develop the *minimal basis* for research on PSMC-ORAMs, upon which more complex and secure protocols could be later developed. Therefore, we imposed particular restrictions on the challenge phase, and changed the second learning phase of the game, in a way that allows us to define the client-to-client access pattern hiding for a PSMC-ORAM in an intuitive and realistic way. We do this with the following variation of the security game from Definition 35:

Definition 37 ($\mathsf{Game}^{\mathtt{Client\text{-}IND\text{-}CQA}}_{\mathtt{A},\mathtt{ORAM}}(\lambda)$)**.**
Let $\mathtt{PSMC\text{-}ORAM} = (\mathtt{PSMC\text{-}ORAM.Init}, \mathtt{PSMC\text{-}ORAM.Access})$ *be a Partial Sharing Multi-Client ORAM construction,* λ *a security parameter and* $\mathtt{A}$ *an ORAM adversary. The computational indistinguishability of client access patterns game under adaptive chosen query attack* $\mathsf{Game}^{\mathtt{Client\text{-}IND\text{-}CQA}}_{\mathtt{A},\mathtt{ORAM}}(\lambda)$ *is played between a challenger* $\mathtt{C}$ *and the adversary* $\mathtt{A}$*. The adversary is a client of the ORAM, and the challenger is a simulator who can simulate any client present on the ORAM except for* $\mathtt{A}$*. The game proceeds as follows:*

1. $\mathtt{A}$ *chooses* $N \leq \tilde{N}$ *and* $K \in \mathbb{N}$*;*
2. $(\mathtt{Client}_1, \cdots, \mathtt{Client}_K, \mathtt{S}) \leftarrow \mathtt{PSMC\text{-}ORAM.Init}(\lambda, (K-1)N)$*, except for some* $\mathtt{Client}_u$ *who is the adversary;*
3. *First CQA learning phase: for* $i = 1, \ldots, q_1 \in \mathbb{N}$*,* $\mathtt{A}$ *repeats (adaptively) the following:*
   a) $\mathtt{A}$ *runs a data request* $\mathtt{mcdr}$ *on his datablocks, or*
   b) $\mathtt{A}$ *does the following:*
      i. $\mathtt{A}$ *chooses a data request* $\mathtt{mcdr}_i$*;*
      ii. $\mathtt{C}$ *executes* $\mathtt{PSMC\text{-}ORAM.Access}$ *on* $\mathtt{mcdr}_i$*;*
      iii. $\mathtt{A}$ *receives* $\mathtt{ap}(\mathtt{mcdr}_i)$*;*
4. *Challenge phase:* $\mathtt{A}$ *chooses two data requests* $\mathtt{mcdr}^0$ *and* $\mathtt{mcdr}^1$ *that include* $\mathtt{Read}$ *or* $\mathtt{Write}$ *operations on datablocks that* $\mathtt{A}$ *does not have access to, and* $\mathtt{Share}$ *or* $\mathtt{Revoke}$ *operations that do not result in* $\mathtt{A}$ *gaining or losing access to datablocks;*
5. $\mathtt{C}$ *flips a random secret bit* $b \xleftarrow{\$} \{0,1\}$ *and executes* $\mathtt{PSMC\text{-}ORAM.Access}$ *on* $\mathtt{mcdr}^b$*;*
6. $\mathtt{A}$ *receives* $\mathtt{ap}(\mathtt{mcdr}^b)$*;*
7. *Second CQA learning phase: for* $j = 1, \ldots, q_2 \in \mathbb{N}$*,* $\mathtt{A}$ *repeats (adaptively) the following:*

a) $\mathtt{A}$ *runs a data request* $\mathtt{mcdr}$ *on his datablocks, or*

b) $\mathtt{A}$ *does the following:*

i. $\mathtt{A}$ *chooses a data request* $\mathtt{mcdr}_i$ *that does not include* `Share` *or* `Revoke` *operation on the datablocks asked in the challenge phase;*

ii. $\mathtt{C}$ *executes* `PSMC-ORAM.Access` *on* $\mathtt{mcdr}_i$*;*

iii. $\mathtt{A}$ *receives* $\mathtt{ap}(\mathtt{mcdr}_i)$*;*

8. $\mathtt{A}$ *outputs a bit* b'.

$\mathtt{A}$ *wins the game if and only if* $b = b'$.

Definition 38 (Client-to-Client Privacy)**.**
We say that a PSMC-ORAM protocol provides Client-to-Server privacy (i.e., hides the access patterns of a client against other clients), if for every PPT PSMC-ORAM adversary $\mathtt{A}$ *the following holds:*

$$\left|\mathrm{Prob}\left[\mathsf{Game}^{\mathit{Client\text{-}IND\text{-}CQA}}_{\mathtt{A},\mathtt{ORAM}}(\lambda)=1\right]-\frac{1}{2}\right|\leq \mathit{negl}(n).$$

6.4. Blurry-ORAM, or the first PSMC-ORAM

In our construction, we consider K clients that store their data on an remote server. Every client's data is partitioned in N datablocks of fixed size B, and each datablock is identified by a unique identifier id. Following the model from Chapter 4, we consider a datablock as a tuple $(\mathsf{id}, \mathtt{dat})$, with id being a unique identifier and $\mathtt{dat}$ being the actual data. In the following we will abuse this notation, and for ease of presentation denote by id_i^j the datablock with identifier id_i that belongs to client j without referring to the actual data of the datablock, unless it is crucial for the description.

For the security parameter λ and K clients, $\mathtt{PSMC\text{-}ORAM.Init}(\lambda, K)$ allocates enough space on the server in order to store the KN real datablocks and their respective fake datablocks. Using a homomorphic, asymmetric encryption scheme $\mathcal{E}$ that is semantically secure and also IK-CPA secure (according to Definition 5), for each client a public- private- key pair is generated by $\mathcal{E}.\mathtt{KeyGen}(\lambda)$, with which every client encrypts his real and fake datablocks and uploads them to the server, in a classical Path-ORAM of height $\lceil \log N \rceil$ and Z datablocks per node. The resulting K Path-ORAM trees are merged on each node, forming a Path-ORAM of height $\lceil \log N \rceil$ and ZK datablocks per node. At this point, in every node, datablocks of all the clients can be found.

The read operation for a datablock id_i^j is a protocol run between a client $\mathtt{Client}_j$ and the server. It returns the datablock identified by id_i^j or $\perp$, if the datablock was not found (for example, if a client queries for a datablock that belongs to other clients but not to $\mathtt{Client}_j$).

The write operation is a protocol executed between a client $\texttt{Client}_j$ and the server. It is similar to the read operation, as it returns the datablock with identifier id_i^j, but overwrites its contents with new data in case the read operation was successful.

The share operation is a protocol run between the server and clients $\texttt{Client}_i$ and $\texttt{Client}_j$. The goal is to make the datablock with identifier id_u^i, which is accessible by client $\texttt{Client}_i$, also accessible (for read, write, share and revoke) by client $\texttt{Client}_j$. This is done by having $\texttt{Client}_i$ change the private key under which the datablock id_u^i is encrypted and handing over this new key to $\texttt{Client}_j$.

The revoke operation is a protocol run between the server and a client $\texttt{Client}_i$. For a datablock with identifier id that can be accessed by $\texttt{Client}_i$ and $\texttt{Client}_j$, the purpose of this operation is to disallow $\texttt{Client}_j$ from further being able to access (read, write, share or revoke) datablock id. This is again done by changing the key under which datablock id is encrypted, but this time not communicating this change to $\texttt{Client}_j$. Note, that the revoke operation is not recursive, and disallows only one specific client all further access to the datablock; thus, after a revoke operation, all other clients to whom the revoked client granted access in the past, are still allowed to read, write, share and revoke the particular datablock.

After a number of data accesses (that also include datablock sharing), the datablocks found in every node are eventually shuffled, and due to the presence of shared datablocks, it can happen that some clients have more datablocks in a node than others. Thus, we end up with a 'blurred' version of the originally merged client trees. Due to this property, as well as to differentiate our construction from the simple concatenation of Path-ORAM trees, where each client is allocated a *fixed* number of datablocks in every node, we will refer to our construction as 'Blurry-ORAM'.

6.4.1. The Protocol in Detail

Our starting point is the Path-ORAM construction of Stefanov et al. [Ste+13a], which we analysed in Chapter 4. In our setting, we consider K clients, who store their data on a data structure residing on a remote server. Every client stores a maximum number of N real datablocks and their corresponding fakes, just like in Path-ORAM. How the clients' data is laid on the remote data structure is of grave importance: The easiest way to do this, would be to construct a tree with KN leaves and assign every datablock to one of those leaves, as in a Path-ORAM. However, such a solution affects the stash size in a way that renders the underlying Path-ORAM inoperable, due to an exceedingly big stash size. The reason for this is that the client can find in every path fewer datablocks of his in a path (which are the only ones he can change or move), and thus the client is forced to use the stash more often. Assume now that each node of the tree can hold Z datablocks and that all clients' datablocks are uniformly distributed in the tree. Assume further that every client can access only those datablocks that belong to him or are shared with him. Then, in every path, a client can only find on average $Z/K \log(KN)$ datablocks belonging to him, as opposed to $Z \log N$ datablocks that he would find if he had stored his datablocks in a single client Path-ORAM. This means that it will be more difficult for him to put the element he read back into

the path, which will eventually force him to use his stash more often. In fact, this was the first way we structured the Blurry-ORAM tree, and our experiments verified this rapid growth of the stash.

In contrast, in Blurry-ORAM we store the clients' datablocks differently: We let each of the K clients store his N datablocks in a separate binary tree with N leaves, where each node holds ZK datablocks, as can be seen in Figure 6.1. Every datablock (real or fake) is encrypted using the client's public key, but in such a way that the datablock can be re-randomised without knowledge of the owner's public key (using for example the encryption scheme proposed in [Gol+04]). Note here, that in order to achieve Client-to-Server and Client-to-Client privacy, any datablock (real or fake) of one client must be indistinguishable from any other datablock of every other client. To achieve this, we have to use an encryption scheme that is also IK-CPA secure according to Definition 5[2] or in other words guarantees the indistinguishability of ciphertexts under different keys. This way, the property is achieved that all datablocks (real and fake) belonging to a client are indistinguishable for anyone but the client who can decrypt them.

As our construction is Path-ORAM based, it inherits the need of using a stash, since there is a chance that during an access, some datablocks cannot be put back in the downloaded path. For datablocks that belong solely to one client, each client locally maintains a stash (called 'localstash' in the following). The bounds on Path-ORAM stash size apply here. However, it might happen that datablocks shared between clients cannot be written back into the path. For this case we maintain a so-called 'commonstash', which will contain all the encrypted shared datablocks that could not fit into the tree. The size of this commonstash can be upper bounded, as we show in Section 6.4.6. The commonstash is initially filled with fake items, encrypted under a key known to every client but not to the server. This way the server cannot observe if shared datablocks have been added or removed, and thus the commonstash can securely remain stored on the server; each client retrieves this stash before he performs any operation. Furthermore, we use a dedicated private key for the fake datablocks on the commonstash, so that the clients can distinguish between fake and real datablocks in the commonstash.

Unfortunately however, this ability of the clients can be a source of privacy leakage: Once client $\texttt{Client}_i$ notices that after a client's $\texttt{Client}_j$ access, a shared datablock has been moved on the commonstash, $\texttt{Client}_i$ immediately knows that in the last accessed path, all datablocks of $\texttt{Client}_j$ are real. This is because the commonstash is used only if the paths accessed are full of real datablocks. Therefore, we must make sure that the commonstash remains as small as possible. We achieve this by changing the way the Read operation is performed in two ways. First, the client downloads *two* paths per access[3]: One determined by the leaf i (which we will from then on call the 'original' path), and a so-called 'symmetric' path, which is the path leading to the symmetric leaf of i, when considering as symmetry axis the line that cuts the leaf level into two parts of equal size, as depicted in Figure 6.3. This second 'symmetric' path is practically used as extra storage space that lightens the usage of the nodes closer to the root. Secondly, during the eviction, instead of using a queue which is used in the original Path-ORAM (and

[2] One such scheme is the ElGamal encryption scheme [Gam85], which is homomorphic, semantically secure and IK-CPA secure, as we will see in Section 6.6

[3] Adopting directly the eviction from [Shi+11], which also involves reading two paths, would unnecessarily degrade the protocol's performance, since we would have to store smaller ORAMs of size $\log(KN)$ in every node.

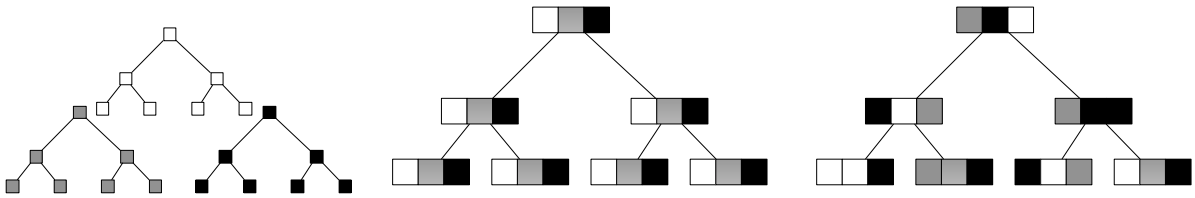


Figure 6.1: Path-ORAMs of three clients, each client assigned different colour for his datablocks (left), concatenated together and forming the initialisation of Blurry-ORAM (middle), and a Blurry-ORAM state after some accesses, leaving the 'blurred' version of the multiple Path-ORAMs (right).

is known to be responsible for the so-called 'stagnant' datablocks) we use a list, which allows us to re-arrange the order into which the datablocks are evicted, thus making sure that first the shared datablocks are evicted, and then all the rest. Having these two heuristics in mind, the read operation proceeds as follows: The client identifies the datablocks he has access to (real, fake and shared). This is done by the client iterating on all the datablocks and checking if a specific condition is fulfilled (for example, by trying to decrypt while using an encryption scheme that returns $\perp$ when decrypting with the wrong key, or by using the properties of homomorphic schemes, as we detail in Section 6.6). By construction, one of these datablocks is guaranteed to be the datablock that the client is looking for. Consequently, the client copies the real and shared datablocks of his in a local list, along with the datablocks in his localstash, and replaces his real, shared and fakes in the paths, with empty placeholders. The client can now use all the empty placeholders in the paths for his eviction. First the client evicts all shared datablocks, trying to store them as deep down in either of the paths as possible. If a shared datablock cannot fit in any of the paths, it replaces a fake datablock in the commonstash. Subsequently, the client evicts in a similar manner those datablocks that are not shared. If there is not enough space in the paths, the localstash is used. The client then fills up the remaining placeholders with fake datablocks, and finally re-randomises all the datablocks in the paths and the commonstash. The paths and the commonstash are then sent back to the server. Indeed, using these ideas, we observed during our experiments that the stash sizes were very small. In particular, the commonstash had been empty during all our experiments, even when we let Z as small as 2.

In our construction, sharing a datablock between clients is straightforward: Suppose client $\texttt{Client}_i$ wants to share datablock id_u with client $\texttt{Client}_j$; $\texttt{Client}_i$ retrieves his datablock id_u (by means of Read operation, which changes also the datablock's path assignment), re-encrypts the datablock with a new key and uploads the datablock to the server. Finally, $\texttt{Client}_i$ hands over to $\texttt{Client}_j$ the *new key* and *the new index of the leaf to which* id_u is mapped through an external communication channel. In a similar way, revocation of access rights is performed: If $\texttt{Client}_i$ wants to revoke access rights of datablock id_u from $\texttt{Client}_j$, $\texttt{Client}_i$ changes the key under which datablock id_u is encrypted (again by reading datablock id_u, and thus changing its path assignment) and informs other clients about the change of key and of path assignment.

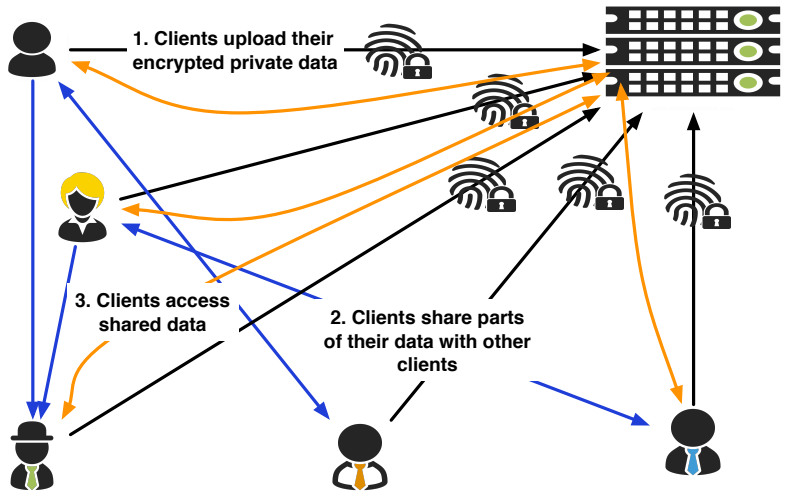


Figure 6.2: Clients upload their encrypted data on a Blurry-ORAM structure (step 1., black arrows), give to other clients access to parts of it (step 2., blue arrows), which the latter access (step 3., orange arrows).

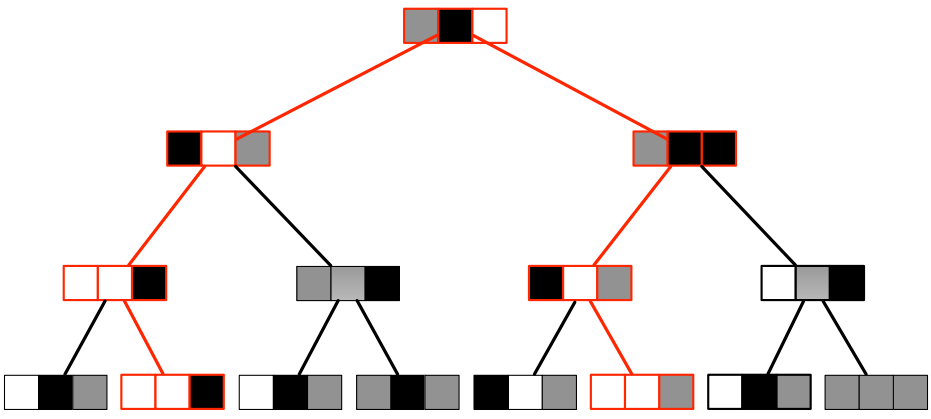


Figure 6.3: Blurry-ORAM's 'Read' operation, downloading the 'original' and the 'symmetric' paths.

6.4.2. Storing the Position Map

In order to save space, in ORAM constructions that use a position map, such as [Ste+13a; Shi+11; SS13b; SSS12], the position map is stored recursively on the server, in smaller ORAMs, $\texttt{ORAM}_1, \ldots, \texttt{ORAM}_k$, where $\texttt{ORAM}_1$ stores the positions of the data and $\texttt{ORAM}_k$ is of constant size. In a PSMC-ORAM that stores a position map recursively, it does not seem possible to avoid the fact that at least $\texttt{ORAM}_2, \ldots, \texttt{ORAM}_k$ must be accessible by all clients. In such a case however, any client can infer that a position has been changed in $\texttt{ORAM}_1$, simply by noticing the changes in $\texttt{ORAM}_2$. This way, a client could trivially break the access pattern privacy of other clients, regardless if they share their data or not. In order to avoid this potential leakage, in Blurry-ORAM we store the clients' position maps in the following way: Every client stores a position map for his own datablocks in a classic Path-ORAM on the server. Similarly, we store a position map in a separate ORAM for every group of shared datablocks to which all the members of the group have access. In order to further prevent the server from knowing whether a client is asking for a shared datablock (which the server can see by observing the position maps being accessed), all clients must access all the position maps for every access they make, even if they cannot decrypt datablocks from certain position maps. In such a case, the client simply downloads a randomly chosen path, re-encrypts it and uploads it back to the server.

6.4.3. Key Management

Suppose $\texttt{Client}_i$ is a client in our system. $\texttt{Client}_i$ has to store a key for the datablocks that he can access exclusively and the key to encrypt/decrypt the fake datablocks in the commonstash. Once $\texttt{Client}_i$ starts sharing datablocks with $\texttt{Client}_j$, $\texttt{Client}_i$ creates a key, under which he can encrypt all the datablocks shared with client $\texttt{Client}_j$, thus forming *groups* of shared datablocks. This way, more clients can share only one key for a whole set of datablocks (for example client $\texttt{Client}_i$ wishes to share all his datablocks with clients $\texttt{Client}_j$ and $\texttt{Client}_u$), and the number of keys stored by every client increases linearly in the amount of groups of shared datablocks.

6.4.4. The Algorithm in Detail

In detail, an access to the Blurry-ORAM is described in Algorithm 6: First, the client finds the leaf to which the datablock he is looking for is currently mapped, and remaps the datablock randomly to a new leaf (lines 2 and 3). The client then downloads the original and its symmetric path, and stores them locally in the lists OriPATH and SymPATH respectively (lines 4 and 5). The client starts processing the paths, beginning with the original path, which he copies to PATH (line 6). The client reads his localstash and the common-stash and copies them in a list $\mathcal{L}$ (lines 8 to 9). For every datablock in the common-stash, the client tries all his keys, in order to decrypt it. On success, the client adds the decrypted datablock to $\mathcal{L}$ (lines 10 to 14). For every datablock in every node in the path, the client tries to decrypt it, using all the keys that he has. If this succeeds, then the client adds the decrypted datablock to $\mathcal{L}$ and marks the

datablock in PATH (lines 15 to 20). Marking the datablock in PATH is crucial, because this way, the client knows where in the PATH his datablocks currently lie, he is therefore able to replace them with other datablocks he has access to, without moving datablocks that belong to other clients.

Once the client has found all the datablocks that belong to him, decrypted them and copied them to $\mathcal{L}$, he scans the list and reads the desired datablock, if the operation was a read or updates the datablock's contents if the operation was a write (lines 23 to 26). Now, if the operation is a share, then if the datablock is already shared, then the client sends the key under which the datablock is encrypted and its current position to $\texttt{Client}_i$; otherwise, the client creates a new key with which he encrypts the datablock and sends this new key and the new position to which the datablock is mapped to $\texttt{Client}_i$ (lines 27 to 32). Revoking access rights for a datablock is done in a similar way, with $\texttt{Client}_i$ changing the encryption key, re-encrypting the datablock and putting it back, as if the operation was a normal write.

Now that the datablock has been found, the client continues with the eviction of the datablocks present in his local list $\mathcal{L}$: As in classical Path-ORAM, the client tries to move as many datablocks as he can closer to the leaf level of the tree. This is done with the function PushBlock, which takes as input a datablock, a path and a leaf and performs the classical Path-ORAM eviction algorithm. If the datablock fits in the input path, PushBlock returns 1, otherwise it returns 0. The client first evicts the shared datablocks. If a datablock does not fit in the path, it is added in the commonstash. This procedure is described in lines 36 to 40. As mentioned earlier, the reason for evicting the shared datablocks first is that at this point there is more available space in the PATH and thus the probability that the shared datablocks actually fit in the tree is higher. This way, the probability of using the commonstash is reduced. Once the shared datablocks are evicted, the client continues with the eviction of all other datablocks (lines 41 to 44).

The original path has now been processed, and the OriPATH is updated with the datablocks from PATH (line 47). Next the client must process the symmetric path (lines 45 to 51). Since the root node of the tree is the only common node between the original and the symmetric path, and during the processing of the original path it has changed, the root of the symmetric path is updated in line 46. The OriPATH is updated with PATH and PATH is emptied. The symmetric path is copied to PATH and is then processed in the same way the OriPATH was previously processed. After this is done, the symmetric path is updated (line 52), both paths and the common-stash are re-randomised (line 53) and finally both paths and the encrypted common-stash are sent back to the server (line 54).

6.4.5. Stash Size

As mentioned above, it is important that the commonstash remains very small. For this reason, the client first evicts all shared datablocks found in the downloaded paths. Clearly however, this does not guarantee that shared datablocks never need to be stored outside the tree, and intuitively, the more shared datablocks exist, the greater the probability that the commonstash will be used.

Suppose that in a Blurry-ORAM with N leaves and K clients, each client shares at most m datablocks. Since every client has a fixed lower bound of buckets that he can use in every node of the Blurry-ORAM,

Algorithm 6: Blurry-ORAM(OP, id, dat, $\texttt{Client}_i$, $\texttt{Client}_j$)

Z: Number of blocks in bucket;
N: Number of client blocks;
λ: Security parameter;
$\texttt{KeyGen}(1^\lambda)$: Key Generation function;
$\texttt{Enc}_{p_k}$: Encryption under public key p_k;
DEC_k: Decryption Algorithm, using key k;
PushBlock(B, PATH, x_1): Path-ORAM eviction algorithm for block B in path $PATH$ and new leaf position x_1;

1 $\mathcal{L} \leftarrow \emptyset$;
/* Find the leaf to which id is mapped in the position map */;
2 x_0 = PositionMap(id);
/* Choose new random leaf */;
3 $x_1 \xleftarrow{R} \mathbb{Z}_N$;
4 OriPATH[] $\leftarrow$ GetPath(x_0);
5 SymPATH[] $\leftarrow$ GetSymmetricPath(x_0);
6 PATH $\leftarrow$ OriPATH;
7 ProcessSymmetricPath = 0;
8 $\mathcal{L} \leftarrow$ ReadLocalStash();
9 C $\leftarrow$ ReadCommonStash();
10 **for** $i = 1$ **to** *C.length()* **do**
11 **for** $j = 1$ **to** KZ **do**
12 **for** k **in** *Keys* **do**
13 **if** $DEC_k(i) \neq \perp$ **then**
14 $\mathcal{L} \leftarrow \mathcal{L} \cup DEC_k(i)$;
15 **for** $i = 1$ **to** *PATH.length()* **do**
16 **for** $j = 1$ **to** KZ **do**
17 **for** k **in** *Keys* **do**
18 **if** $DEC_k(PATH[i][j]) \neq \perp$ **then**
19 $\mathcal{L} \leftarrow \mathcal{L} \cup DEC_k(PATH[i][j])$;
20 Mark($PATH[i][j]$);
21 **for** i **in** $\mathcal{L}$ **do**
22 **if** i.id == id **then**
23 **if** $OP ==$ "R" **then**
24 RetBlock = i;
25 **else if** $OP ==$ "W" **then**
26 $i.data =$ dat;
27 **else if** $OP ==$ "$Share$" **then**
28 **if** i *is not shared* **then**
29 $k \leftarrow \texttt{KeyGen}(1^\lambda)$;
30 $i.data = \texttt{Enc}_k(\texttt{dat})$;
31 Send x_1 to $\texttt{Client}_j$;
32 Send k to $\texttt{Client}_j$;
33 **else if** $OP ==$ "$Revoke$" **then**
34 $k \leftarrow \texttt{KeyGen}(1^\lambda)$;
35 $i.data = \texttt{Enc}_k(\texttt{dat})$;
/* First evict the shared blocks */;
36 **for** i **in** $\mathcal{L}$ **do**
37 **if** i **is** *shared* **then**
38 **if** PushBlock(i, PATH, x_1) == *0* **then**
/* Shared block did not fit into the tree */;
39 C $\leftarrow i$;
40 Remove i from $\mathcal{L}$;
/* Now evict the remaining blocks */;
41 **for** i **in** $\mathcal{L}$ **do**
42 **if** PushBlock(i, PATH, x_1) == *0* **then**
/* Shared block did not fit into the tree */;
43 C $\leftarrow i$;
44 Remove i from $\mathcal{L}$;
45 **if** ProcessSymmetricPath == 0 **then**
46 SymPATH[0] $\leftarrow$ PATH[0];
47 OriPATH $\leftarrow$ PATH;
48 PATH $\leftarrow \emptyset$;
49 PATH $\leftarrow$ SymPATH;
50 ProcessSymmetricPath = 1;
51 Repeat Steps 8 to 44;
52 SymPATH $\leftarrow$ PATH;
53 Rerandomise(OriPATH, SymPATH, C);
54 Upload(OriPATH, SymPATH, C);
55 **return** *RetBlock*

we could simulate every client's data as being stored in a single Path-ORAM, in which the client stores $N + m$ real datablocks in a tree with N leaves. Since during eviction we let the shared datablocks take the place of real datablocks that belong to the client and we first push these shared datablocks into the structure, in essence we treat these m datablocks as the real datablocks and all other datablocks as fakes. Thus, in order to examine the commonstash size, we can simulate a Blurry-ORAM by a Path-ORAM, in which a client stores m real datablocks in a tree with N leaves. Based on the proof of [Ste+13b] we can estimate the probability of using a stash of size $O(\log \log N)$ for $m = \log^2 N$, by following the same argument as in the classical Path-ORAM, and adjusting the number of leaves of the tree. These ideas are summarised in the following lemmata:

Lemma 6.4.5.1.
For a data request sequence α in a Blurry-ORAM with K clients, each having N datablocks and sharing $O(\log^2 N)$ datablocks, with $Z = 5$ buckets per client per node, that uses a commonstash of size R, the probability that the size of the commonstash during a data request sequence α exceeds R during one of the requests is bounded by

$$\text{Prob}[\mathsf{Stash}(\mathit{Blurry\text{-}ORAM}_L^Z)[\alpha] > R] = 14 \cdot (0.6002)^R.$$

Proof. (sketch) Observe that the shared datablocks are a subset of the original N real datablocks and recall that in Blurry-ORAM the shared datablocks are the ones that are evicted first. Thus, the bounds of Path-ORAM's stash analysis from [Ste+13b] apply for the shared datablocks as well. □

Following the argumentation line of [Ste+13b], we have that for $s = O(\log^2 N)$ accesses (i.e., accessing all shared datablocks), the probability that the commonstash grows larger than R is $s \cdot 14 \cdot (0.6002)^R$, which means that for $s = \text{poly}(\log^2 N)$ accesses, the probability that the commonstash will be larger than $O(\log \log N)$ during one of the accesses to the shared datablocks is negligible in N.

6.4.6. Time and Space Requirements

Based on the observations made earlier in this section, we can now analyse the time and space requirements of our protocol. A client that participates in $O(\log N)$ groups needs $O(\log N)$ space for the keys and $O(\log N)$ for the position maps (given the recursive position map storage). The client also needs to store his private stash which follows the bounds provided in [Ste+13a], and is thus limited to $O(\log N)$. Each time the client performs a data request, he downloads two paths of size $O(\log N)$, and the commonstash, which is in the worst case of size $O(\log \log N) \in O(\log N)$. Thus, the amount of space needed during protocol execution for the client is $O(\log N)$.

As far as the computational complexity of the client is concerned, recall that the client has to iterate on his $O(\log N)$ keys for each of the downloaded datablocks found in the paths, thus the computational complexity is $O(\log^2 N)$. Note that since we have restricted the amount of shared datablocks to $O(\log N)$

per client, there is a total $O(K \log N) \in O(\log N)$ position maps from each of which a client will read and re-randomise a path during a query.

Suppose that a client shares $N-1$ datablocks of his with K clients. Instead of holding a different key for every group, the client can share all the common datablocks with all the clients, using only one key, and create smaller position maps only for the datablocks that are not shared with all of the clients. Thus, each smaller position map will not exceed a size of $O(\log N)$, which means that for every query, the client will have to dedicate $O(\log^2 N \log(\log N))$ time for the recursive position map accesses.

6.5. Security Analysis

In order to show that a PSMC-ORAM is secure, one has to show that it achieves client-to-server and client-to-client privacy, by means of Definitions 35 and 37 respectively. Client-to-server privacy for Blurry-ORAM is shown under the assumption that a homomorphic, semantically secure and IK-CPA (according to Definition 5) encryption scheme is used. In addition to these properties, however, in order to show client-to-client privacy for Blurry-ORAM, one has to also assume that the commonstash is used with negligible probability. We show these properties in the following.

Proposition 9.
For a public key encryption scheme that is homomorphic, IND-CPA and IK-CPA secure, Blurry-ORAM with at least two clients achieves client-to-server Privacy against a semi-honest adversarial server according to Definition 35.

Proof. Suppose that there exists a PPT adversary $\mathtt{A}$ that wins the Server-IND-CQA game on a Blurry-ORAM, with non-negligible advantage δ. We show how to construct a PPT algorithm $\mathtt{B}$, that breaks either the semantic security, or the key indistinguishability property of the encryption scheme used in Blurry-ORAM. To do this, suppose that $\mathtt{B}$ plays the IK-CPA and the IND-CPA game against a challenger $\mathtt{C}_0$ and a challenger $\mathtt{C}_1$ respectively, and wins if he wins any one of them. Let $\mathcal{E} = (\mathtt{KeyGen}(n), \mathtt{Enc}_{pk}(\cdot), \mathtt{Dec}_{sk}(\cdot))$ be the public key encryption scheme used in Blurry-ORAM, with $(pk, sk) \leftarrow \mathtt{KeyGen}(\lambda)$ a private/public key pair produced for the security parameter λ.

$\mathtt{C}_0$ runs $\mathtt{KeyGen}(\lambda)$, issues $(\mathtt{pk}_i, \mathtt{sk}_i)$ for $i = 2, \ldots, K-1$ and sends $\{\mathtt{pk}_i\}_{i=2}^{K-1}$ to $\mathtt{B}$. Similarily, $\mathtt{C}_1$ runs $\mathtt{KeyGen}(\lambda)$, issues $(\mathtt{pk}_0, \mathtt{sk}_0)$, $(\mathtt{pk}_1, \mathtt{sk}_1)$, and sends $\mathtt{pk}_0$ and $\mathtt{pk}_1$ to $\mathtt{B}$. $\mathtt{B}$ simulates the Blurry-ORAM server and runs $\mathtt{Blurry\text{-}ORAM.Init}(\lambda, N, K)$ in order to create the private/public key pairs for all the clients present on the Blurry-ORAM. $\mathtt{B}$ assigns each $\mathtt{pk}_i$ to $\mathtt{Client}_i$ for the public keys he got from $\mathtt{C}_0$ and $\mathtt{C}_1$, and $\mathtt{B}$ keeps two versions of Blurry-ORAM: One which will be encrypted and one plain.

On a data request $\mathtt{mcdr}$from $\mathtt{A}$, the $\mathtt{View}(\mathtt{mcdr})$ is formed by $\mathtt{B}$ as follows:

1. $\mathtt{B}$ adds to the $\mathtt{View}$ the two paths from the encrypted $\mathtt{Blurry\text{-}ORAM}$ version.

2. B now runs the eviction algorithm on the plain Blurry-ORAM version, encrypts every client's datablock found with the respective client's key, and updates the encrypted version.
3. B finally encrypts the commonstash and adds it to the View, along with the encrypted updated paths from the previous step.

The result of the above operation is a view that A can process.

In the challenge phase, B receives from A a tuple of two data request sequences $(\mathtt{mcdr}_0, \mathtt{mcdr}_1)$. B copies the plain Blurry-ORAM, including all the position maps and stashes of the clients, in a Blurry-ORAM-COPY and runs both of the data request sequences, on each Blurry-ORAM (for example $\mathtt{mcdr}_0$ on Blurry-ORAM and $\mathtt{mcdr}_1$ on Blurry-ORAM-COPY) as earlier in the pre-challenge phase. B first chooses a random bit $\tilde{b}$; if $\tilde{b} = 0$, B will play against $\mathtt{C}_0$ (the IND-CPA challenger), else he plays against $\mathtt{C}_1$ (the IK-CPA challenger). Now B chooses a random bit b^* and selects the data request sequence $\mathtt{mcdr}_{b^*}$ and the Blurry-ORAM yielded thereof (either the Blurry-ORAM or the Blurry-ORAM-COPY). In the two plain views, B finds the positions where the unencrypted views differ in the datablocks of client $\mathtt{Client}_u$, picks the corresponding (plain) data $(m_1^0, m_2^0, \ldots, m_q^0)$, $(m_1^1, m_2^1, \ldots, m_q^1)$ corresponding to $\mathtt{mcdr}_0$ and $\mathtt{mcdr}_1$ respectively and forms two pairs of challenges: the IND-CPA challenge, $\left(\{m_i^0\}_{i=1}^q, \{m_i^1\}_{i=1}^q\right)$ along with $\mathtt{pk}_u$ and the IK-CPA challenge, $\left(\{m_i^{b^*}\}_{i=1}^q, \mathtt{pk}_0, \mathtt{pk}_1\right)$. B sends the two challenges to $\mathtt{C}_0$ and $\mathtt{C}_1$ who respond by issuing $\{\mathtt{Enc}_{pk_u}(m_i^b)\}_{i=1}^q$, and $\{\mathtt{Enc}_{pk_b}(m_i^{b^*})\}_{i=1}^q$ respectively. In $\mathtt{DRS}_{b^*}$, B now injects the appropriate ciphertext: If B chose to play against $\mathtt{C}_0$, he injects $\mathtt{Enc}_{pk_u}(m_j^b)$ and if he chose to play against $\mathtt{C}_1$, he injects $\mathtt{Enc}_{pk_b}(m_j^{b^*})$. Then B forms the rest of $\mathtt{View}(\mathtt{DRS}_{b^*})$ as he did in the pre-challenge phase, and passes the result to A.

The post-challenge phase continues exactly as the pre-challenge phase with data requests issued from A. At the end of this phase, A outputs a bit b', which B uses to output (b', d) if he had chosen to play against IND-CPA, or (d, b'), if he had chosen to play against IK-CPA, where d is a random bit.

Observe now, that if B chooses to play against IND-CPA, and if $b = b^*$, then A gets a well formed view and thus by assumption, A wins with non-negligible advantage δ. In a similar manner, if B plays against IK-CPA A gets a well formed view and thus by assumption, A wins with non-negligible advantage δ. Suppose now, that if $\tilde{b} = 0$ and $b^* \neq b$, A outputs 0 with probability α_0 and outputs 1 with probability $1-\alpha_0$, while if $\tilde{b} = 1$ and $b^* \neq b$, A outputs 0 with probability α_1 and outputs 1 with probability $1-\alpha_1$. Then B's total winning probability (i.e. breaking IND-CPA or IK-CPA) is

$$\mathrm{Prob}\left[{}^{\mathrm{Bwins}}_{\text{IND-CPA}} \text{ or } {}^{\mathrm{Bwins}}_{\text{IK-CPA}}\right] =$$

$$\frac{1}{8}\left(\frac{1}{2}+\delta+\alpha_0+(1-\alpha_0)+\frac{1}{2}+\delta+(1-\alpha_1)+\frac{1}{2}+\delta+\alpha_1+\frac{1}{2}+\delta\right) \geq \frac{1}{2}+\frac{\delta}{2q},$$

which means that with non-negligible advantage, B breaks either the semantic security or the key indistinguishability property of the encryprion scheme, which is a contradiction. □

Proving that our construction achieves access pattern privacy against clients is more involved and is done by showing that Blurry-ORAM satisfies Definition 38. To do this, however, we must first make sure that the commonstash is rarely used. Indeed, as we saw in Lemma 6.4.5.1, the size of the commonstash is very small with high probability.

Proposition 10.
Assume that in Blurry-ORAM the server is semi-honest, and that the clients are semi-honest and not colluding with each other. Assume further that the commonstash is used with some negligible probability ϵ in Blurry-ORAM's security parameter λ, and an asymmetric, semantically secure, homomorphic and IK-CPA secure encryption scheme $\mathcal{E}$ is used. Then Blurry-ORAM achieves client-to-client privacy according to Definition 37

The main proof idea is that we assume the existence of a PPT adversary that breaks the client-to-client privacy, by winning the IND-CQA game with non-negligible advantage, and we then construct an algorithm that, by carefully crafting the views the adversary creates, as well as the views he is given through oracle access to other clients' data requests, he implants the IND-CPA or the IK-CPA challenge in the IND-CQA challenge, and can break the encryption scheme's semantic security, or its IK-CPA property, thus yielding a contradiction.

Proof. Suppose that there exists a PPT adversary $\mathtt{A}$, that wins the Client-IND-CQA game on a Blurry-ORAM with non-negligible advantage δ_1, when the commonstash is used and wins the Client-IND-CQA game with some non-negligible advantage δ_2, when the commonstash is not used. We show how to construct a PPT algorithm $\mathtt{B}$, that breaks either the semantic security or the key indistinguishability property of the encryption scheme used in Blurry-ORAM. To do this, suppose that $\mathtt{B}$ plays the IND-CPA and the IK-CPA games against two respective challengers $\mathtt{C}_0$ and $\mathtt{C}_1$, and wins if he wins against any of them. $\mathtt{A}$ will be a client in the Blurry-ORAM, who will attack the access patterns of other clients present on the construction. $\mathtt{B}$ will simulate all the other clients, as well as the server, and will use $\mathtt{A}$ in order to win either the IND-CPA or the IK-CPA game.

Let $\mathcal{E} = (\mathtt{KeyGen}(\lambda), \mathtt{Enc}_{\mathtt{pk}}(\cdot), \mathtt{Dec}_{\mathtt{sk}}(\cdot))$ be the public key encryption scheme used in Blurry-ORAM, with $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathtt{KeyGen}(\lambda)$ a private/public key pair produced for the security parameter λ. $\mathtt{C}_0$ and $\mathtt{C}_1$ run $\mathtt{KeyGen}(\lambda)$, get $(\mathtt{cpk}, \mathtt{csk})$ and send $\mathtt{csk}$ to $\mathtt{B}$. $\mathtt{B}$ simulates the Blurry-ORAM server and runs the $\mathtt{Init}$algorithm in order to create the private/public key pairs for all clients present on the Blurry-ORAM, except for $\mathtt{A}$ (who is also a client in Blurry-ORAM). For one of the clients that $\mathtt{B}$ simulates, say client $\mathtt{Client}_i$, $\mathtt{B}$ uses the public key $\mathtt{cpk}$ that he got from his challenger $\mathtt{C}_0$, instead of creating a fresh one. $\mathtt{B}$ keeps the Blurry-ORAM in plaintext, except for the blocks that $\mathtt{A}$ has access to and are not shared by him, which $\mathtt{B}$ keeps encrypted on the Blurry-ORAM.

Whenever $\mathtt{B}$ gets a data request from $\mathtt{A}$, for a block that belongs to $\mathtt{A}$, $\mathtt{B}$ does the following:

1. $\mathtt{B}$ finds the paths that $\mathtt{A}$ asks for. The encrypted blocks found in P, $\mathtt{B}$ leaves as they are (since these are the blocks that only $\mathtt{A}$ has access to and thus $\mathtt{B}$ never sees in plain), and every block in plain,

B encrypts using each client's encryption key since everything in the view must be encrypted. B encrypts the blocks in the path that A asks for using for each block the encryption key that corresponds to the block's owner (including the blocks that belong to client $\texttt{Client}_i$).

2. To every node of the paths, B adds A's encrypted blocks that were found on that node. These (encrypted) blocks, B discards from his data structure. Now the paths are encrypted and A can proccess it.

3. B sends the encrypted paths and the commonstash to A, who runs his eviction algorithm and sends the updated paths and the commonstash back to B.

4. B now decrypts all the blocks that he can in the paths and the commonstash he received and updates the corresponding blocks in the plain Blurry-ORAM version. This way any possible changes that A made on shared datablocks, are considered. The datablocks that could not be decrypted belong to either $\texttt{Client}_i$ or A. Since the datablocks accessible only by $\texttt{Client}_i$ cannot have been moved in any of the paths during A's eviction, B replaces these encryptions with the plaintext blocks of $\texttt{Client}_i$ from the paths of his plain Blurry-ORAM.

5. B discards the version of the paths on his Blurry-ORAM structure, and sets in their place the newly updated paths, which contain the datablocks of all clients in plain (including the ones that A shares with other clients), except the datablocks that only A has access to, which are encrypted.

   To every node of the updated paths of the plain Blurry-ORAM construction, B adds the datablocks from the corresponding node of the path received from A, that B could not decrypt (and thus the nodes that belonged to A).

Whenever B gets an 'oracle' data request from A on datablocks that do not belong to A, B runs the Blurry-ORAM protocol and produces an unencrypted view as follows:

1. B adds to the unencrypted view the appropriate paths (which we will call the downloaded paths).

2. B updates the appropriate position map and runs the eviction algorithm, using the commonstash if necessary.

3. B runs the eviction on the plain datablocks and adds the updated paths (which we will call the uploaded paths) and the commonstash to the unencrypted view.

After the eviction is done, B creates the view, by encrypting it as follows: For every element found in the view, B (who knows to which client each block belongs) encrypts it, using the appropriate client's public key. Finally, B adds to the downloaded paths A's encrypted datablocks that were found on the paths, and to the uploaded paths, a re-randomisation of these encrypted blocks. The result of this operation is a view that A can process.

Note that in forming the view, only the downloaded, the uploaded paths and the commonstash are used. This is because the position maps for all the blocks that are not shared or belong to A are kept encrypted on the server (a role being played by B here) and that all other stashes are locally stored by every client.

In the challenge phase, B receives from A, a tuple of two data request sequences $(\mathtt{DRS}_0, \mathtt{DRS}_1)$. B copies the Blurry-ORAM, including all the position maps, the stashes of the clients and the commonstash, in a Blurry-ORAM-COPY and runs both of the data request sequences, on each Blurry-ORAM (for example $\mathtt{DRS}_0$ on Blurry-ORAM and $\mathtt{DRS}_1$ on Blurry-ORAM-COPY) as earlier in the pre-challenge phase. B first chooses a random bit $\tilde{b}$; if $\tilde{b} = 0$, B will play the IND-CPA game, else he plays the IK-CPA game. Now B chooses a random bit b^* and selects the data request sequence $\mathtt{DRS}_{b^*}$ and the Blurry-ORAM yielded thereof (either the Blurry-ORAM or the Blurry-ORAM-COPY). In the two unencrypted views there will be at least one position in the data-blocks of client $\mathtt{Client}_i$, where the unencrypted views will differ. B finds the first position where the two plain views differ, picks all the corresponding (plain) data after that position, $\{m_j^0\}_{j=1}^q$ corresponding to $\mathtt{DRS}_0$ and $\{m_j^1\}_{j=1}^q$ corresponding to $\mathtt{DRS}_1$, and forms two pairs of challenges: the IND-CPA challenge, $\left(\{m_j^0\}_{j=1}^q,\ \{m_j^1\}_{j=1}^q\right)$ and the IK-CPA challenge, $\left(\{m_j^{b^*}\}_{j=1}^q,\ cpk,\ pk\right)$, where cpk is client's $\mathtt{Client}_i$ public key and pk is the public key of another client. B sends the two challenges to $\mathtt{C}_0$ and to $\mathtt{C}_1$. The IND-CPA challenger responds by encrypting $\{m_j^b\}_{j=1}^q$ under cpk for a random bit choice b, and the IK-CPA challenger responds by encrypting $m_j^{b^*}$ under pk or cpk. $\mathtt{C}_0$ and $\mathtt{C}_1$ send their responses back to B.

B now encrypts the plain view corresponding to $\mathtt{DRS}_{b^*}$, in the same way he did in the pre-challenge phase, with the only difference, that he implants the encryptions he was given from $\mathtt{C}_0$ or from $\mathtt{C}_1$, based on his former choice of $\tilde{b}$.

The post-challenge phase continues exactly as the pre-challenge phase with data requests issued from A, which however cannot include share and revoke operations on the blocks requested during the challenge phase. At the end of this phase, A outputs a bit b'. B then outputs (b', d) if he had chosen to play against IND-CPA, or (d, b'), if he had chosen to play against IK-CPA, where d is a random bit.

Suppose now that the probability of using the commonstash while performing $\mathtt{DRS}_{b^*}$ is

$$\text{Prob}\left[\begin{smallmatrix}\text{B used commonstash}\\ \text{for } \mathtt{DRS}_{b^*}\end{smallmatrix}\right] = \epsilon$$

and observe that if B chooses to play against IND-CPA, and if $b = b^*$, then A gets a well formed view and thus by assumption, A wins with non-negligible advantage δ_1, thus

$$\text{Prob}\left[\mathtt{A}\text{ wins } \middle|\ \left(\text{commonstash used and } b = b^*\right)\right] = \frac{1}{2} + \delta_2$$

and

$$\text{Prob}\left[\mathtt{A}\text{ wins } \middle|\ \left(\text{commonistash not used and } b = b^*\right)\right] = \frac{1}{2} + \delta_1.$$

The same applies if B chooses to play against IK-CPA. Suppose further that if $\tilde{b} = 0$ and $b \neq b^*$, A outputs 0 with probability α_0 and 1 with probability $1 - \alpha_0$, and if $\tilde{b} = 1$ and $b \neq b^*$, A outputs 0 with probability α_1 and 1 with probability α_1. Then B's total winning probability (i.e. breaking IND-CPA or IK-CPA) is

$$\text{Prob}\left[{}^{\text{Bwins}}_{\text{IND-CPA}} \text{ or } {}^{\text{Bwins}}_{\text{IK-CPA}}\right] \geq \frac{1}{4}\left(2 + 2\delta_1 + \epsilon(\delta_2 - \delta_1)\right)$$

However, we have by assumption that $\epsilon = \text{negl}(\lambda)$, thus

$$\text{Prob}\left[{}^{\text{Bwins}}_{\text{IND-CPA}} \text{ or } {}^{\text{Bwins}}_{\text{IK-CPA}}\right] \geq \frac{1}{2} + \frac{\delta_1}{2q},$$

which means that with non-negligible advantage, B breaks either the semantic security or the key indistinguishability property of the encryprion scheme, which is a contradiction. □

6.6. A Concrete Instantiation

Recalling the description of the Blurry-ORAM architecture (see Section 6.4), we note that the main cryptographic building block upon Blurry-ORAM is based, is the encryption scheme, that has to fulfil the following properties. Firstly, the encryption scheme has to be semantically secure. This is a typical requirement for any ORAM construction, as the server should not be able to distinguish between encryptions of the same datablock. Secondly, the encryption scheme has to be asymmetric. Although this is not a typical requirement in most ORAM constructions (in fact one tries to avoid usage of heavy Public Key Infrastructure (PKI)), it seems that in a multi-client setting such a requirement is a necessity. The reason for this, is the fact that every client should be in position to re-randomise any datablock he sees without knowing the secret key with which the datablock is encrypted. Thirdly the encryption scheme has to provide indistinguishability of keys under which the datablocks are encrypted, a property achieved by IK-CPA secure encryption schemes. The reason for this, as we saw, is that the server (or other clients) should not be in position to associate datablocks with specific clients. Lastly, for a practical instantiation of the protocol, the encryption scheme has to allow re-randomisation of ciphertexts without knowledge of the public key. Otherwise, the efficiency of the scheme would be severely hindered by the amount of public keys that every client should store, and by the effort needed for every client to associate each datablock with the respective public key under which it is encrypted.

An encryption scheme that supports all the requirements stated above, is the ElGamal encryption scheme (see Section 3.2.1). First the ElGamal cryptosystem is asymmetric, semantically secure and IK-CPA secure as shown in the following.

Proposition 11.

The ElGamal encryption scheme is IK-CPA secure

Proof. Theorem 3.1 of [Bel+01b]. □

Achieving the last requirement for the encryption scheme (i.e., re-randomisation without knowledge of the public key) is done by taking advantage of the cryptosystem's homomorphic properties and combining them with ideas from [Gol+04]. For a given datablock block, we store as its encryption the tuple (c_0, c_1, c_2, c_3), where $(c_0, c_1) = \mathtt{Enc}_k(\mathsf{block})$ and $(c_2, c_3) = \mathtt{Enc}_k(1)$ for an ElGamal public key k. Re-randomisation is done as follows: First one re-randomises the (c_2, c_3) part by choosing a random value r and obtaining $(c_2', c_3') = (c_2^r, c_3^r)$, and then re-randomising (c_0, c_1) is simply done by setting $(c_0' = c_0 c_2',\ c_1' = c_1 c_3')$.

It is important to note, that the way we described above to store the encryption of a datablock, solves simultaneously two additional issues. For one, suppose that we have an ElGamal encrypted message m, $\mathtt{Enc}_k(m) = (g^r, mg^{rs})$ for a private-, public-key pair (s, k) and a group generator g. Observe that decrypting using a wrong private key s', still yields a group element, since for any $r' \in \langle g \rangle$, there always exists a unique $z \in \langle g \rangle$, such that $z = mg^{rs}g^{rs'}$. This means that a client decrypting with the wrong key, will not be in the position to tell that he got a wrong decryption. However, using the above structure of the encrypted datablock, a client that has a secret key s, simply needs to verify if $c_2^s = c_3$ in order to know if he has the correct decryption key or not. At the same time, this is a far more efficient way for a client to check if a datablock belongs to him or not, since this procedure involves only one exponentiation instead of a complete and costly decryption.

Chapter 7

Application to Genomic Studies

Now that we have proposed ORAM constructions that allow multiple clients to store and process their data, we need to show that these constructions can be used in realistic data-intensive problems.

Can the proposed ORAM architectures of Proxy-ORAM (Chapter 5) and Blurry-ORAM (Chapter 6) be deployed in realistic applications?

We answer the above question in a positive way by using our two architectures for privacy preserving processing of genomic data. We couple Proxy-ORAM with Yao's garbled circuit solution for Secure Two-Party Computation (Section 3.3.3), and show that this way Proxy-ORAM can be used for a variety of genomic tests, offering a high degree of privacy, since the data retrieved and processed is always encrypted and only the end result is announced. However, using Proxy-ORAM for Genome Wide Association Studies (GWAS) cannot guarantee access pattern privacy between the clients. For this case, we use Blurry-ORAM in a multi-client setting, where clients store their encrypted DNA data on a remote server and give partial access to their data to multiple investigators. This way, the latter can retrieve DNA data of multiple clients and use it in a privacy preserving manner as the investigator sees only the DNA parts that are of interest for him.

Published Content

The results from this chapter appeared in [Kar+14] and in [KPK17]. Both protocols have been implemented and tested by myself, while the definition of the experimental framework was done together with Kay Hammacher, Andreas Peter and Stefan Katzenbeisser in [Kar+14], and with Andreas Peter and Stefan Katzenbeisser in [KPK17].

7.1. Privacy-Preserving Genomic Tests

Personalised medicine based on genomic data will result in a need to store genetic data as part of a patient's electronic health record. However, the great benefits of biostatistical analysis and large collections

of genomic data for public health will put pressure on the traditional understanding of privacy. Technically, the major roadblock for this development is already removed: the costs to sequence an individual's genome dropped well below $400. Thus, the great benefits and ever sinking costs lead us to expect a sharp increase in the volume of stored genomic data in the coming years.

Unfortunately, the availability of a large set of genetic data incurs great privacy problems; genetic data can arguably be seen as one of the most sensitive forms of medical data. Furthermore, our knowledge on the human genome increases over time; at present, it is impossible to estimate the future consequences in case a breach of genomic data occurs. Thus, genomic databases should be protected with strong PETs early on. The most promising approach to protect genomic data is the use of cryptographic techniques from secure computation such as Secure Two Party Computation, since these techniques provide strong cryptographic security, both during storage and processing. In this approach, genomic data is stored in encrypted form and the evaluation is performed directly on encrypted data. This both protects the raw genomic data and allows to control the types of queries that can be performed.

However, STC techniques, unless they touch all the data, do not provide privacy during accessing the outsourced data, which in the scenario of genomic processing can be a source for significant private information leakage. Indeed, when running genomic tests, only a small part of the outsourced data is used, and thus merely observing which parts of it are being accessed can lead to knowledge of the test being run. Using Proxy-ORAM, we can synergistically combine ORAM techniques with secure two-party computation solutions in order to offer privacy preserving computations on outsourced, fully sequenced DNA, while at the same time offering full query flexibility. Our starting point is storing a client's fully sequenced DNA in small datablocks and outsourcing it to a remote server, in a way that the client's access patterns are hidden. Using secure computation techniques on the retrieved DNA datablocks, the computation can be performed in an oblivious manner. Decoupling the data retrieval and computation process, we obtain full flexibility to adapt to future queries.

As we have seen in Chapter 5, Proxy-ORAM employs two separate servers (the 'cloud' and 'proxy') which jointly operate an ORAM that (in our scenario here) will store encrypted genetic data. A client can subsequently authorise a different party, called 'investigator' to perform a query on the data in two steps: In the first step, the investigator retrieves the required encrypted genetic data from the ORAM in a way that does not require the client to be constantly online. Here, in a second step we will couple Proxy-ORAM with STC techniques, so that for a specific computation on the retrieved datablocks, the investigator can obtain the result of the computation by running any secure computation protocol between the cloud and the proxy. This solution provides the flexibility to perform any genetic test securely. Further, this solution can be extended to a setting where the remote server stores data of multiple users in one or more different ORAMs, thus allowing the investigator to compute on different users' encrypted data stored on the server's database. Finally, we demonstrate the practical feasibility of the approach by implementing three analysis techniques, operating on (simulated) fully sequenced genomes. Note here however, that in such a scenario using Proxy-ORAM as the underlying ORAM architecture, no client-to-client privacy as in Definition 38 can be achieved.

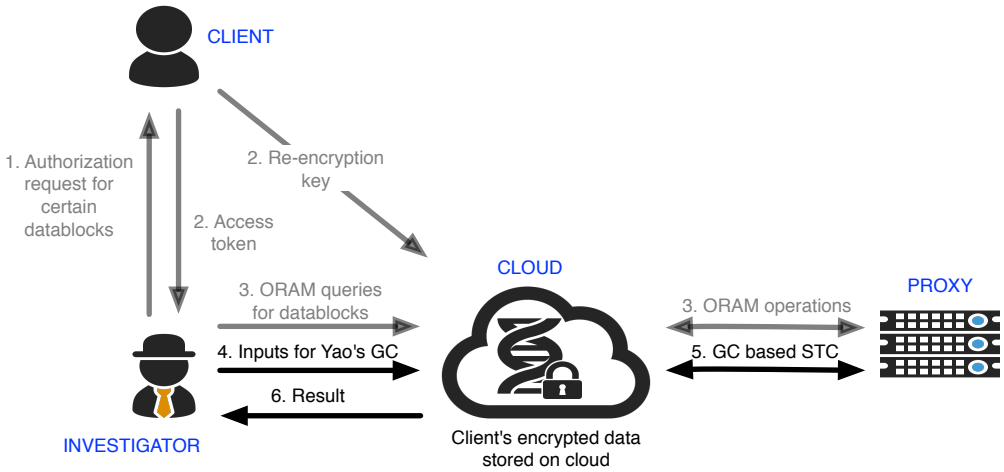


Figure 7.1: Overview of the Proxy-ORAM architecture coupled with Yao's Garbled Circuit solution. In steps 1 and 2, the client authorises the investigator to access his encrypted data. In step 3, the investigator retrieves the client's datablocks using the ORAM interface. In steps 4 and 5, the investigator asks the cloud and the proxy to run Yao's garbled circuit protocol on the datablocks retrieved from step 3, and in step 6 the investigator attains the result of the computation.

Upon acquisition of genetic data, a client uploads his DNA in small encrypted blocks to the cloud. Subsequent access to the data is illustrated in Figure 7.1: Whenever the investigator wants to access certain DNA datablocks of a client, he first asks the client for authorisation (steps 1–2) and retrieves the required blocks from the cloud in a private manner (step 3) through the Proxy-ORAM protocol. Once all data is retrieved, the investigator asks the cloud and the proxy to jointly compute a certain function (e.g., a medical test or study) on the retrieved encrypted blocks, in a secure and private way (steps 4–5). To do this, the cloud and the proxy run Yao's garbled circuit protocol, with the (encrypted and blinded) inputs and circuit provided by the investigator. Throughout these steps, all private data from the client remains encrypted and only the result of the computation is revealed to the investigator (step 6). Neither the cloud nor the proxy learns anything about the stored data, including the investigator's access patterns (except for the number of blocks accessed). This architecture can be generalised in a straightforward way to tests that operate on multiple genomes: in the first step, all required encrypted data blocks are fetched from the cloud given the consent of all involved users. Note that in such a multi-client setting, DNA data of multiple clients can be stored in one ORAM database; alternatively one ORAM can be used per client.

The retrieved DNA blocks are homomorphically encrypted and subsequently converted to shares, which are given to the cloud and the proxy, along with a Boolean circuit representation of the functionality the investigator wants to evaluate. This transformation allows full flexibility of queries as the cloud and the proxy can run Yao's garbled circuit for Secure Two-Party Computation for any desired functionality.

Let $\{\mathtt{Enc}_{\mathtt{pk}}(\mathtt{dat}_i)\}_{i=1}^{k}$ be the blocks that the investigator retrieved from the cloud during this first step. In order to evaluate a specific functionality f on these blocks, the investigator first converts the encryption into shares as follows: he blinds the retrieved encrypted data with a random value and sends it to the proxy, while the blinding values are sent to the cloud. Recall that in Proxy-ORAM, the proxy and the

cloud are both semi honest and do not collude. Recall further, that the proxy sees either blinded or proxy re-encrypted versions of encrypted datablocks. Thus, we can give the client's private key to the proxy without affecting Proxy-ORAM's security. In such a case, the proxy can decrypt the obtained blocks and is left with *blinded* versions of the required data items. Since the cloud now has the blindings, this means that the proxy and the cloud have an additive linear secret sharing of the data contained in the blocks. Subsequently, the investigator creates the circuit corresponding to the functionality f he wants to compute (which needs to include the necessary data unblinding step) and sends it to the proxy who performs the circuit garbling. The cloud and the proxy can now evaluate the desired function on the secret shared blocks together, using any garbled circuit framework. Finally, the result of the computation is returned to the investigator.

7.1.1. Genetic Tests Using our Framework

As we have seen in Chapter 2.2, a sequenced human genome is a set of approximately 3.3 billion characters, out of which only around 1% seem to be relevant for humans. This specific part of the genome is typically stored in the form of SNPs, which are the positions in the sequenced genome of an individual which differ from what is known as the reference genome – and what is believed to be a representative example of a human genome. Once a client's SNPs have been determined, they are stored in biobanks, alongside those of other individuals, and can be used for various tests. We illustrate this general procedure, as well as the flexibility and potential of our framework by three specific use cases which are often performed in the area of genetic analysis.

Pattern Matching in SNPs

In our architecture it is straightforward to search for a specific mutation in a block of sequenced DNA; this conforms to the present practice of finding markers via the SNPs. Here, the investigator first retrieves the encrypted block that contains the SNP to be analysed. Subsequently, the investigator runs a simple comparison protocol, illustrated in Figure 7.2 that compares the SNP part of the block to the desired mutation. Technically, the circuit for this operation needs to unblind the retrieved encrypted datablock, extract some nucleotides and perform the comparison operation. We used the compiler from [Hol+12]. Representing each nucleotide by two bits in a DNA block, we produced the corresponding circuit, which consists of 20,490 gates. Using a framework like [Bel+13], which evaluates a gate per 7.25 ns, this circuit can be evaluated in approximately 5270 µs on a Gigabit network.

DNA Fingerprints in Forensics

Usage of genomic data in criminal forensics has become a wide-spread tool [Roe13]. It is based on the identification of STRs which are highly polymorphic regions of short repeated sequences of DNA – each around four nucleotides long. Typically, investigators focus on several loci[1] in the genome, looking for several STRs. Then, the number of repeated copies is extracted and used as a 'description vector' for an

[1] For instance, the US Combined DNA Index System (CODIS) maintained by the FBI uses 13 such loci.

individual or a probe found at a crime scene. We stress that there is conclusive evidence that STRs reveal information about family members and kinship [BBL06].

Note that this procedure is related but substantially different from the above SNP pattern matching for the following reasons: Firstly, the output is a vector of integers representing the amount of copies of individual STRs, and secondly the loci are not necessarily in vicinity but scattered through the *entire* genome.

Initialising the parameters of our system as will be discussed in Section 7.1.2 and thus storing 128 character long DNA blocks, we implemented the above approach using [Hol+12]. The resulting circuit that outputs the 'description vector' is of size 1,697,280 gates, which again using the framework from [Bel+13] can be evaluated in roughly 1 second.

Statistical Queries

In this setting, the sequenced genome of multiple clients is examined, and associations between specific parts of the human genome and various diseases are made. To do this, a biostatistician (the investigator) examines parts of the genome of multiple DNA samples that are suspected to be associated with a specific disease. Further, the investigator consults a table indicating whether a client suffers from the disease or not, and thus can extract the probability that a particular DNA region (or SNP) is correlated with the disease.

To this end, one can perform statistical analyses: Consider the events $A \in \{0,1\}$ stating that the patient suffers ($A = 1$) or does not suffer ($A = 0$) from a particular disease, and $B_i^p \in \{0,1\}$ stating that a distinctive pattern p occurs in DNA block indexed by i. Note, that p and i do not necessarily contain all possible and thus combinatorially many patterns of nucleotides or DNA fragments, but rather a small and well understood subset.

A biostatistician knows the disease classifications and wants to train a statistical model for future prediction of disease prevalence. He can thus compute the prior probabilities $\text{Prob}(B_i^p|A)$ for a given set of stored genomes – taken as a training set. Furthermore, he can compute the prior probability distribution $\text{Prob}(A)$ on the prevalence of the disease in the sample. Using our architecture, the investigator can obtain all probabilities $\text{Prob}(B_i^p)$ for $B_i^p \in \{0,1\}$ – without revealing the individual genomes. Now, using Bayesian estimation, a medical practitioner who was provided with the biostatistician's model can diagnose a patient by computing the probability $\text{Prob}(A|B_i^p) = \frac{\text{Prob}(B_i^p|A)\cdot\text{Prob}(A)}{\text{Prob}(B_i^p)}$ given a genome of a patient. Note, that updates on the model can be performed as well: whenever new genomes enter the pool of genomic data in the cloud (preferably many), the biostatistician can use Bayesian updating to accommodate the new data and modify the effective model $\text{Prob}(A|B_i^p)$.

In order to estimate the complexity of this approach, we implemented the computation of the above-mentioned probabilities on 1000 retrieved encrypted DNA data blocks using fixed point arithmetic with 10 bit precision. The resulting circuit consists of 21,989 gates for one block, out of which 21,590 gates were used to perform the unblinding of the 2048 bit long blinded block. Since the circuit creation scales

```
#define k // position of the character in the datablock
          // INPUT_A_x: The blinded encrypted DNA datablock given to the cloud
          // INPUT_B_y: The blinding value given to the proxy
void findCharacter(INPUT_A_x, INPUT_B_y){
    int TEST;
    int z;
    int recoveredBlock;
    recoveredBlock = (INPUT_A_x + INPUT_B_y);
    z = (recoveredBlock >> 2k) && 3;
    if (z == TEST)
        return 0;
    else
        return 1;
}
```

Figure 7.2: Search for specific value in an encrypted datablock

linearly, the resulting circuit for 1000 DNA blocks has a size of about 21,000,000 gates, which can be evaluated in 5.4 seconds using the framework from [Bel+13] over a Gigabit network. Note that this number amounts for the time required to perform computations on the retrieved blocks; timings for the data retrieval part are given in the next section.

7.1.2. Proxy-ORAM Experimental Setup and Results

As we have seen, once the datablocks have been retrieved from the server in a privacy-preserving manner, performing any function evaluation on them using Yao's garbled circuit method for secure computation can be done in a very efficient way. Therefore, observing that the real bottleneck of our framework's efficiency is Proxy-ORAM, we focused our experimental evaluation on the performance of Proxy-ORAM when initiated with a fully sequenced genome.

We follow the recommendations of Ecrypt II [Blu], which are slightly more conservative than those of NIST. Thus we choose for the ElGamal group a safe prime $p = 2qr + 1$ of length 1024 bits with q and r primes and set the discrete logarithm key of size q, to 160 bits. In order then for the CvHP hash function to be compatible with the ElGamal encryption we choose the element α (as mentioned in Section 3.2.1) to be a generator of the subgroup $\mathbb{Z}_q^*$ and the generator β to be a multiple of α. The block ids are drawn uniformly and at random from $\mathbb{Z}_q^*$, making sure that every block (fake and real) is assigned a different id. We choose the BCP keys to be of size 4096 bits, thus having the underlying prime factors of size 1024 bits. Regarding our Bloom filters and in order to minimise the disk seeks during the queries, we set the number of Bloom filter keys to 5, randomly chosen from $\mathbb{Z}_q^*$. For each level, the Bloom filter is of size 50 times the size of the level, and we use 5 keys for the hash function, thus achieving a theoretical false positive ratio smaller than 2^{-18}, using the results from [CRJ10] [2].

[2] Note here that the probability of a false positive does not affect the correct retrieval of a block, as pointed out in Sections 5.2.2 and 5.4

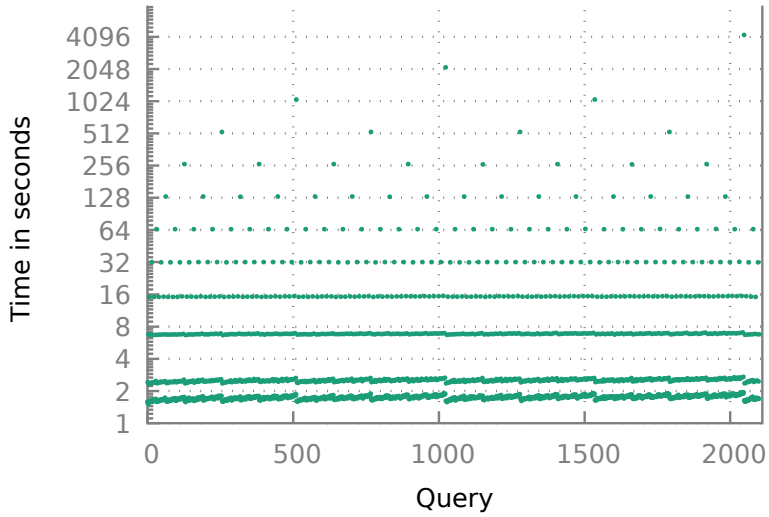


Figure 7.3: Average time (10 repetitions) of 2100 queries on a database with 2^{25} real packets.

We implemented our mechanism on an *IBM System x3550 M3* server equiped with *32GB of RAM* and two *Intel Xeon X5650 2.67GHz* processors running *Ubuntu Server 11.10*. Our system was implemented in C++ and compiled using *g++* from the *GNU Compiler Collection* version 4.6.1. For the cryptographic operations, we used the *Crypto++* library in version *5.6.1*. We used MySQL version 5.1.69 as the storage backend.

Needing only two bits to encode a nucleotide, in order to store the entire genome's approximately 3.3 billion characters, we would need approximately 2^{32} bits. Using the BCP parameters described above which support plaintext blocks of size at most 2048 bits, we could store a fully sequenced human genome in less than 2^{25}=33,554,432 blocks of 128 characters each. We simulated the client's DNA, by storing random numbers representing the various DNA blocks. We argue that in a person's lifetime no more than 2^{25} queries over his DNA are expected to be performed, thus we implemented the optimization described in Section 5.5 regarding the generation of the last level's Bloom filter: We did not create a Bloom filter for the ORAM's last level during the initial data upload.

Our tests showed that the system's bottleneck is its initialization: we measured the initialization of databases of various sizes and observed that it scales linearly: a 2^{17}=131,072 blocks database needed approximately 2 hours and 36 minutes, a 2^{24}=16,777,216 blocks database needed approximately 7 days and the largest database, corresponding to a fully sequenced genome, holding all 2^{25} blocks, needed approximately 14 days to be populated. Nevertheless, as this is an operation that is only performed once during the lifetime of a client, these durations seem justified.

In order to test the efficiency of the block retrieval from the database, we set up two databases of different size: a database containing 2^{17} blocks and the large database containing 2^{25} blocks. In both databases we performed 10 rounds of 2100 queries. In Figure 7.3, we show the average timings for the 10 rounds of queries on the large database: We see that the majority of queries run in less than 100 seconds. Note that this scenario resembles the case of storing a full genome. The peaks in this graph occur whenever a

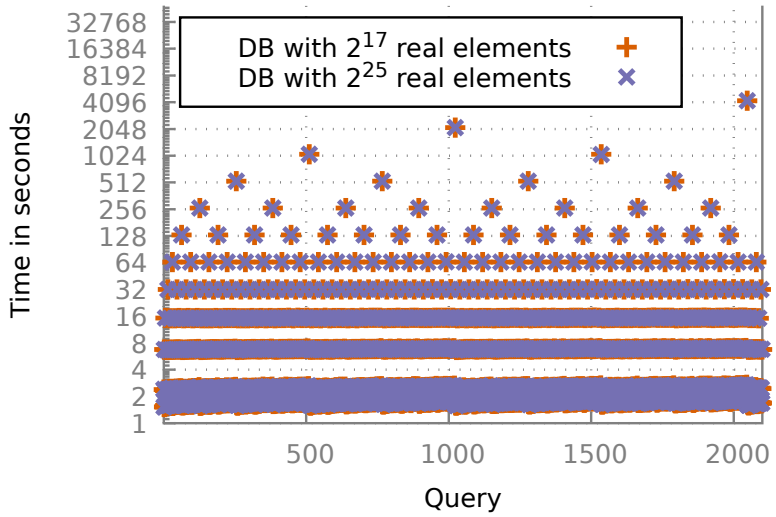


Figure 7.4: Average time (10 repetitions) of 2100 queries on a database the two databases holding 2^{17} and 2^{25} real packets respectively.

reshuffle is performed. Observe the periodicity of the data: The time needed for a reshuffle of a higher level amounts to twice that of the previous level. However since the elapsed time between two reshuffles increases exponentially, a low amortised time of approximately 12.39 s seconds per query is achieved. In Figure 7.4 we compare the average time needed for 10 repetitions of the 2100 queries in the two databases. From the results we see that the query time is roughly independent of the size of the database and depends only on the number of the queries performed so far. This is expected, as at any given time, not all the ORAM levels are full.

To measure the traffic exchanged between the parties, we performed 2100 queries locally on the loopback interface. We used *iptables* for traffic accounting, thus the numbers also include all TCP and IPv4 headers and not just the application layer payload. Figure 7.5 depicts the summed traffic between the cloud and the proxy, which shows certain 'jumps'. These occur whenever a reshuffle is performed and as in the time measurement results, each jump's height is twice that of the previous one, with an exponential amount of queries being performed from one jump to the next. During the 2048-th query, 318 MB are sent from the proxy to the cloud and on average only 1.202 MB are transferred per query between the proxy and the cloud. Figure 7.6 shows the traffic between the investigator and the other parties, with an average of 133 kB per query. Note that (like in the case of the time measurements) the traffic is only affected by the number of queries performed and not by the number of items in the database. The results clearly indicate that the investigator is almost not burdened by the query, as required by the problem domain.

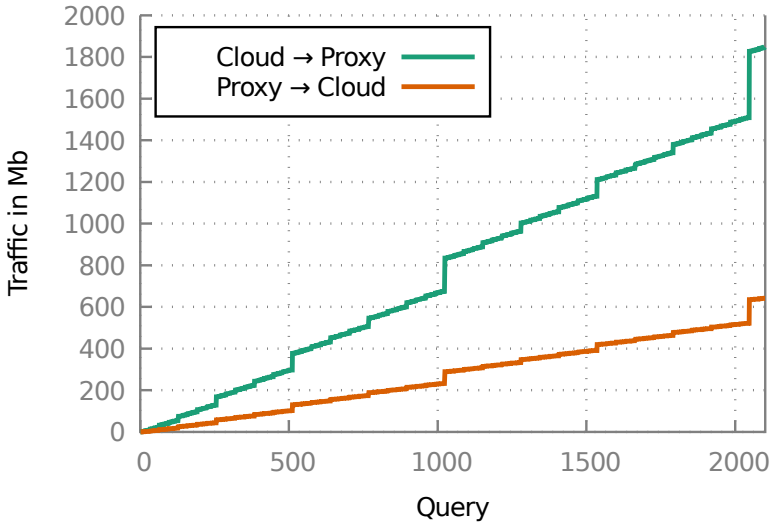


Figure 7.5: Traffic between the cloud and proxy.

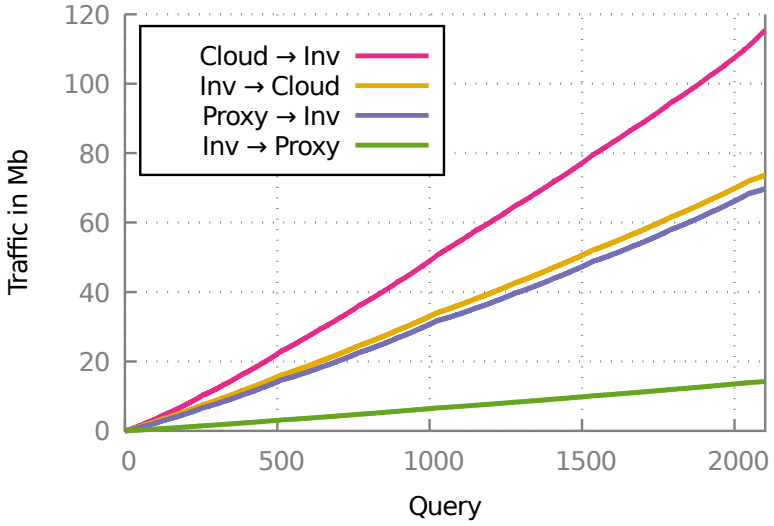


Figure 7.6: Traffic between the cloud and the investigator, and between the proxy and the investigator.

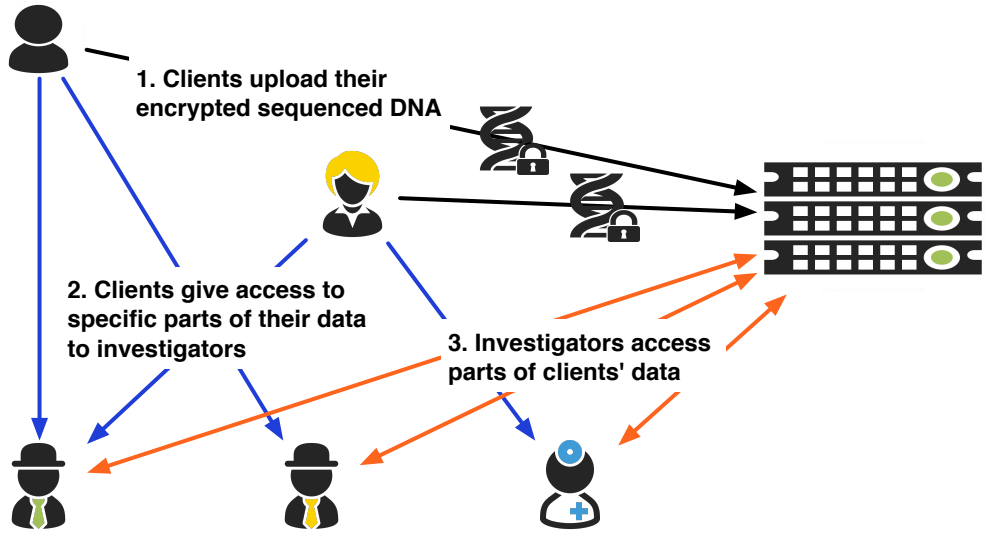


Figure 7.7: Clients upload their encrypted genome on a Blurry-ORAM structure (step 1.), give to investigators access to parts of it (step 2.), which the latter access (step 3.)

7.2. Privacy-Preserving GWAS

In all but the last of the applications described in Section 7.1.2, we have assumed that *one* client stores his encrypted genetic data on the remote server. In the case of the 'Statistical Queries', we assumed the existence of multiple clients without taking into consideration the information leakage that can potentially appear in such a setting, as we have seen in Chapter 6. In fact, Proxy-ORAM was not designed with such security guarantees in mind. Therefore, in order to be able to perform 'Statistical Queries' with higher levels of client privacy, it seems straightforward to turn to Blurry-ORAM.

Typically, what we described earlier as 'Statistical Queries', falls into the category of Genome Wide Association Studies (GWAS), where an investigator examines whether a set of genetic variants in a number of clients is associated with a specific disease. Due to the large data sizes involved, storing a client's genomic data is typically done by storing the client's SNPs. Once all the clients' SNPs are stored on the remote server (the biobank), the clients give selectively access to a subset of their data, to multiple investigators (aka the biostatisticians), in the form of what we described as shared blocks in Chapter 6. Thus, in this case, we do not have only multiple clients, but also multiple investigators; an overview of this scheme is depicted in Figure 7.7. As we have seen in Blurry-ORAM's security analysis, Blurry-ORAM guarantees that the existence of the shared blocks does not leak any information about the rest of the blocks, while (as was the case in Proxy-ORAM as well) the client and the investigator are not required to be constantly (or even simultaneously) online.

7.2.1. Experimental Setup

We implemented Blurry-ORAM on a virtual machine running Ubuntu Version 16.10 with 8 cores and 16GB of RAM, hosted on a 2x Xeon E5 2620v2 server with 12 Cores (24HT) and 64GB of RAM, using VMWare ESXi 6 for the virtualization. For the client we used a desktop PC equipped with an AMD FX(tm)-8350 Eight-Core Processor and 24GB of RAM, running on Ubuntu Desktop Version 16.04. As backend on the server, we used MySQL version 5.7.16. The code was compiled using $g++$ version 6.2.0, for the cryptographic backend we used the OpenSSL library, version 1.0.2, and the experiments were ran on a 1Gbit LAN network. Note that using as many cores as possible is crucial in boosting the efficiency of our construction: After the client has identified the blocks he can decrypt, he can perform the eviction in parallel, with re-randomising all other blocks using all available cores.

7.2.2. Experiments

Using the techniques proposed in [BWB09], one can efficiently store a patient's genome by using roughly 2^{17} SNPs. Using an elliptic curve over a prime field $G(p)$ with p of size 256 bits (cf. section 3.2), we can map a SNP to a single point of the elliptic curve. Typically for elliptic curve cryptography, mapping a plaintext to a point of the elliptic curve is not a trivial task. This is usually solved by concatenating random noise to the plaintext so that it can be mapped to a point of the elliptic curve. Indeed, doing this and using 16 bits of randomness for every block, we were able to map all the plaintexts we had, to points of the elliptic curve. Observe also, that for every SNP, we needed maximally 17 bits to represent the identifier and 16 bits of randomness to do the mapping to the elliptic curve. This left us with 223 bits, which provide adequate space to store the SNP, using the techniques described in [BWB09].

With the above in mind, we stored in our simulations the SNPs of 100 clients, resulting in a database holding a total of 2^{23} blocks (2^{17} SNPs per client, enough to hold a human genome stored in the form of SNPs as described in [BWB09]). We then allowed multiple investigators to access specific parts of the stored genomic data. We assume that every client distributed a different key to every investigator, so that neither other clients, nor investigators could learn anything about blocks they would not have access to. Figure 7.8 shows the performance of our construction: we performed 10 rounds of 1000 queries and measured the time needed for every query, depending on the number of keys a client has access to. The time needed for each query is affected by the number of investigators present (since decryption is attempted with all the client's keys), and varies from 6.48 s on average, when only one investigator is present, to 63.01 s when 100 investigators are present.

In a similar setting, we varied the number of clients storing their data (again we use 2^{17} SNPs per client), while 100 investigators were present, each one of them using a different key. We performed 10 rounds of 1000 queries. The results are presented in Figure 7.9, where we observe that when 10 clients were present, the average time for a query was 6.9 s, when 50 clients were present the average time was 31.40 s, and when 100 clients were present the average time for a query was 63.01 s.

In Figure 7.10 we examined how the number of blocks affects the performance of our construction. We instantiated our construction with 50 clients and 10 investigators (i.e., 10 keys per client) and stored 2^{15}, 2^{16} and 2^{17} blocks per client. We performed again 10 rounds of 1000 queries and measured an average query time of 6.92 s, 7.42 s and 7.71 s, when each client stored 2^{15}, 2^{16} and 2^{17} blocks, respectively.

As we have seen in the security analysis of Blurry-ORAM (see Section 6.5), the occupancy of the commonstash plays a very important role for Client-to-Client privacy. To experimentally assess the size of both the local- and commonstash, we performed 10 rounds of 2^{17} queries (i.e., access of all blocks) in the database of 100 clients with 2^{17} blocks, 1000 keys each, and set $Z = 2$, i.e., 2 blocks per client per node. We examined the sizes of the localstash and the commonstash. The results are shown in Figure 7.11, where we see that the commonstash was *never* used. On the other hand, the localstash attained once a maximum occupancy of 20 blocks, which is less than the theoretical approximations of the original Path-ORAM scheme [Ste+13b].

The previous experiments focused on storing all SNPs, which store only the positions where a human's genome differs from the reference genome. Having in mind that it still remains to be answered what information is hidden in the 90% of an organism's genome (which today seems irrelevant for GWAS), we the consider the case where one would choose to store an individual's whole genome. Using elliptic curves as described above, we can store the approximately 3×10^9 characters, using 2^{23} blocks per client, with each block storing approximately 128 characters[3]. We ran a series of 10 rounds of 1000 queries in a database with 2, 4 and 8 clients that share 10 blocks. The results are shown in Figure 7.12, where we see that the average query time was 50 s, 55 s, 60 s respectively. In a similar manner, we ran 10 rounds of 1000 queries in a database storing the whole genome of 8 clients, that share 10, 50 and 100 blocks and the results. Figure 7.13 shows that the average query time was 4.11 s, 6.98 s and 11.26 s respectively, thus yielding a practical solution.

[3] Note that since the genome's alphabet consists only of the four letters A, T, G, C, we only need 2 bits to represent each letter of the alphabet.

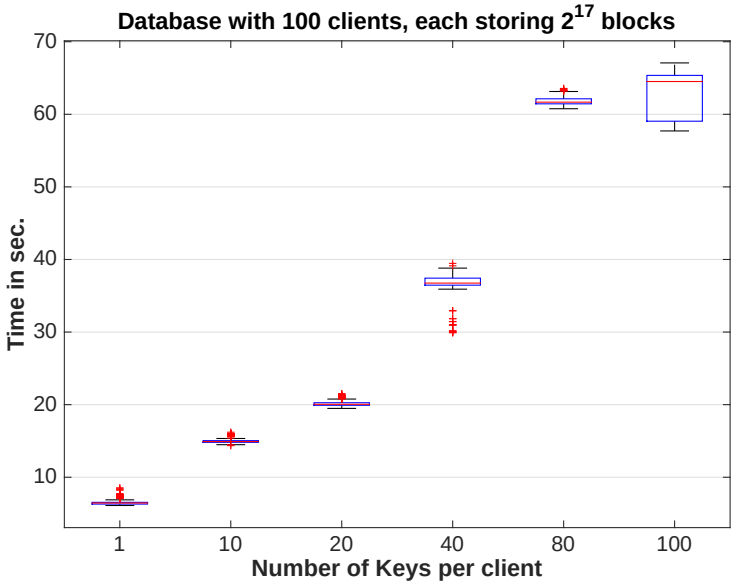


Figure 7.8: Performance of Blurry-ORAM, when varying the number of keys in the scenario of storing the SNPs.

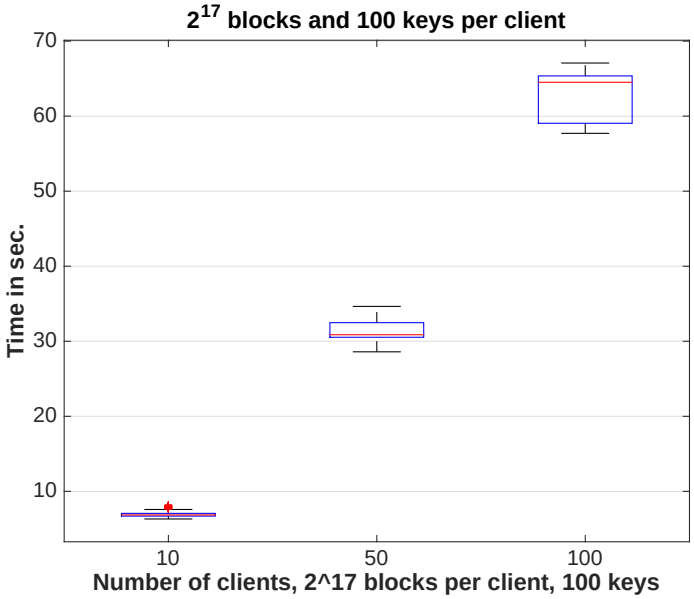


Figure 7.9: Performance of Blurry-ORAM, when varying the number the number of clients in the scenario of storing the SNPs.

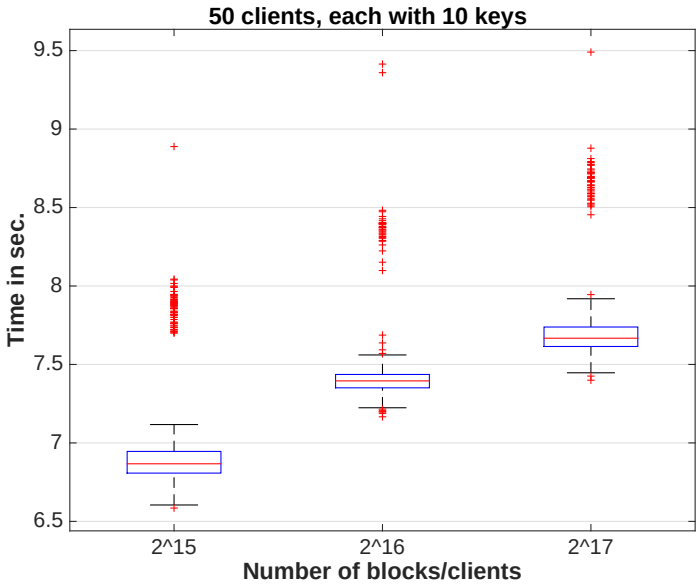


Figure 7.10: Performance of Blurry-ORAM when varying the number of blocks per client.

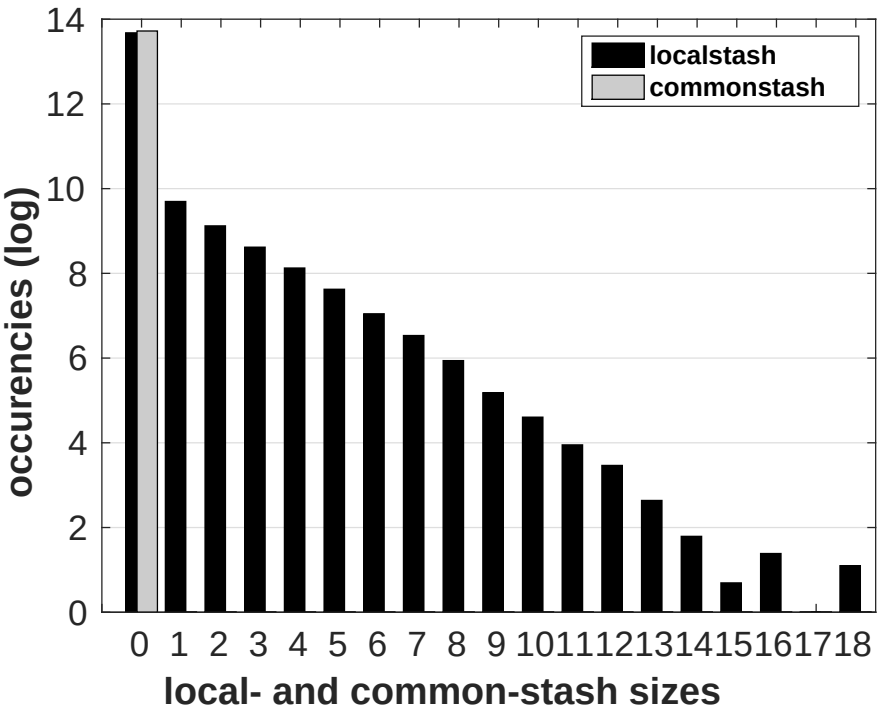


Figure 7.11: Sizes of the local- and commonstash for $Z = 2$ with each client sharing 1000 blocks and storing 2^{17} blocks.

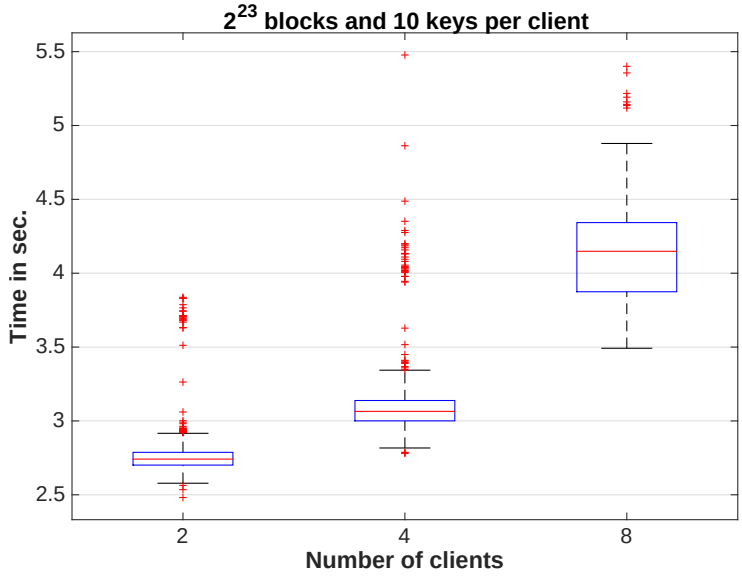


Figure 7.12: Blurry-ORAM performance, when varying the number of clients. The experiment used $Z = 2$, and allowed each client to store 2^{23} blocks.

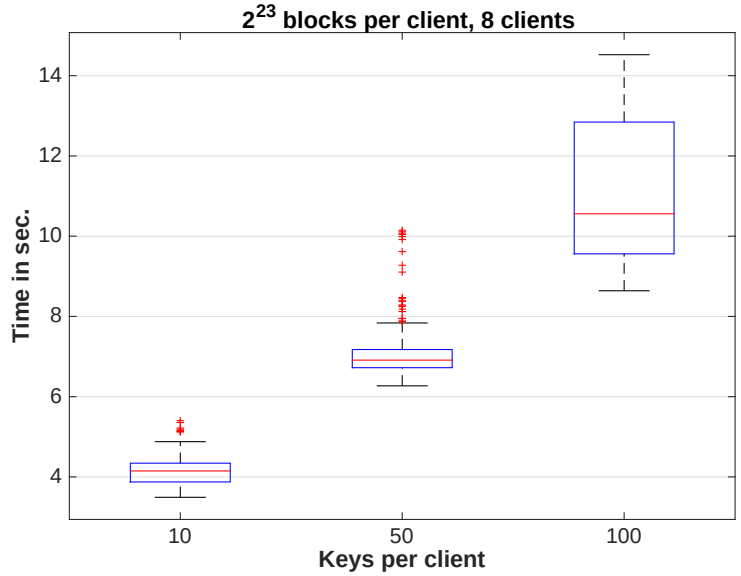


Figure 7.13: Blurry-ORAM performance, when varying the number of keys per client. The experiment used $Z = 2$, and allowed each client to store 2^{23} blocks.

Chapter 8

Conclusion

In this work we addressed the problem of constructing PETs for data intensive problems in a two-fold way. Our first goal was to construct new ORAM solutions that allow for a multitude of entities to participate in the protocol. Our first construction, Proxy-ORAM, presented in Chapter 5, allowed a client to store encrypted data on a remote untrusted server. Subsequently, the client could access his data in a privacy preserving way, and give temporary access to it to a third party, who would see only encrypted data during the course of the protocol. Furthermore, the client was not involved in any of the communication and computationally intense steps of ORAM.

In our second construction, Blurry-ORAM, presented in Chapter 6, we introduced the notion of PSMC-ORAMs. These are ORAM architectures that allow multiple clients to store their encrypted data on a remote server, and at the same time share with each other parts of their data. We identified the privacy leakage that can occur in such settings, and provided the necessary theoretical framework that allows for the security evaluation of such schemes. Based on one of the currently most efficient ORAM constructions, we developed the first PSMC-ORAM, and evaluated its security.

Arguing about the security guarantees of such complex ORAM architectures would not have been an easy task without the new versatile ORAM security framework which we developed in Chapter 4. Our new framework closes the gap between the austere ORAM formalisation of Goldreich and Ostrovsky, and the high-level security proofs that have predominated the existing ORAM works. The potential of our new framework is shown not only in the development of rigorous security proofs for existing schemes, but mainly in its agility to extend these schemes into new and more complex ones, such as the newly introduced PSMC-ORAM framework.

In order to further argue about the practicability of our proposed solutions, we tested our new constructions on a data intensive problem that we believe will play a focal role in the way privacy and freedom will further develop in our society. Coupling Proxy-ORAM with ideas from STC, we tested the applicability of our solution for privacy preserving genome processing. In another setting, yet still regarding genomic processing, we used Blurry-ORAM to instantiate a scenario where multiple clients store their genomic data on remote servers and share with each other (or with other parties such as biostaticians and other biobanks) parts of their data.

Our results have shown that our solutions are practical. Yet, there is much space for further research. In Proxy-ORAM, the existence of the Bloom filter affects the soundness and the security of the protocol in a sympathetic way: A small (and thus a more efficient instantiation of the protocol) Bloom filter increases the probability of a false positive. Given that the datablocks that the investigator sees at the end are encrypted, this can have a serious effect in the correctness of the computation's result. At the same time, a high false positive probability means that the ORAM guarantees can be easily compromised. To avoid these obstacles, we had to employ a Bloom filter of large size, which meant a serious compromise of efficiency. Therefore, finding a substitution of the Bloom filter is a major and interesting challenge for this solution.

Blurry-ORAM is the first functional PSMC-ORAM. Knowing that the databases storing genomic data for multiple clients would quickly grow, we based our construction on the most efficient ORAM construction to-date. This came at the cost of having to use the so-called commonstash, which in turn can seriously affect the protocol's security guarantees. Therefore, building a PSMC-ORAM that avoids the use of such a data structure, while relying on other ORAM constructions such as distributed ORAM is an open and interesting problem.

It is still an open problem, how our constructions can be further refined, in order to address the problem of client anonymity. It is also of great interest (that will have an important impact on the applicability of our solutions) to extend them in a setting that guarantees security against malicious servers. Further, an important question that needs to be answered in future work, is how the PSMC-ORAM model can be modified in order to provide security against malicious and colluding clients.

In this work, our main ORAM focus was in the client/ server model, and for our developed solutions we were motivated by the problem of secure genomic processing. Due to the sensitivity of genomic data, it is of great importance to provide long-term security. One way of doing this, is by showing that our constructions are post- or even fully-quantum secure. In [GKK17], we showed that it is possible to extend ORAM security in both post- and fully-quantum settings. This result, gives us a strong intuition on how our ORAM constructions can be extended and proven secure in such settings. Further, it is interesting to examine if Yao's Garbled Circuit solution can also be extended in the post-quantum setting, and which of its numerous optimisations can still be proven secure against post-quantum adversaries.

Bibliography

[Abr08] Daniel H. Abrahams Brett S.and Geschwind. “Advances in autism genetics: on the threshold of a new neurobiology”. In: *Nat Rev Genet* 9.5 (May 2008), pp. 341–355. ISSN: 1471-0056. DOI: 10.1038/nrg2346 (cit. on p. 2).

[Adr+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella Béguelin, and Paul Zimmermann. “Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015 (cit. on p. 13).

[App+10] Benny Applebaum, Haakon Ringberg, Michael J. Freedman, Matthew Caesar, and Jennifer Rexford. “Collaborative, Privacy-Preserving Data Aggregation at Scale”. In: *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010. Proceedings*. Vol. 6205. Lecture Notes in Computer Science. Springer, 2010, pp. 56–74 (cit. on p. 38).

[Ate+06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. “Improved proxy re-encryption schemes with applications to secure distributed storage”. In: *ACM Trans. Inf. Syst. Secur.* 9.1 (2006), pp. 1–30 (cit. on p. 50).

[Ayd+13] Erman Ayday, Jean Louis Raisaro, Urs Hengartner, Adam Molyneaux, and Jean-Pierre Hubaux. “Privacy-Preserving Processing of Raw Genomic Data”. In: *Data Privacy Management and Autonomous Spontaneous Security - 8th International Workshop, DPM 2013, and 6th International Workshop, SETOP 2013, Revised Selected Papers*. 2013, pp. 133–147 (cit. on p. 12).

[ARH13] Erman Ayday, Jean Louis Raisaro, and Jean-Pierre Hubaux. “Privacy-Enhancing Technologies for Medical Tests Using Genomic Data”. In: *20th Annual Network and Distributed System Security Symposium, NDSS 2013*. 2013 (cit. on p. 12).

[Bac+16] Michael Backes, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. “Anonymous RAM”. In: *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Proceedings, Part I*. Vol. 9878. Lecture Notes in Computer Science. Springer, 2016, pp. 344–362 (cit. on pp. 10, 64, 65).

[Bal+11a] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. “Countering GATTACA: efficient and secure testing of fully-sequenced human genomes”. In: *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011*. 2011, pp. 691–702 (cit. on p. 12).

[Bal+11b] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. “Countering GATTACA: efficient and secure testing of fully-sequenced human genomes”. In: *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011*. ACM, 2011, pp. 691–702 (cit. on p. 12).

[Bel+01a] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. “Key-Privacy in Public-Key Encryption”. In: *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*. Vol. 2248. Lecture Notes in Computer Science. Springer, 2001 (cit. on p. 15).

[Bel+01b] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. *Key-Privacy in Public-Key Encryption (full version)*. 2001. URL: `https://cseweb.ucsd.edu/~mihir/papers/anonenc.pdf` (cit. on p. 87).

[Bel+97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. “A Concrete Security Treatment of Symmetric Encryption”. In: *38th Annual Symposium on Foundations of Computer Science, FOCS ’97*. IEEE Computer Society, 1997, pp. 394–403 (cit. on pp. 29, 30).

[Bel+13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. “Efficient Garbling from a Fixed-Key Blockcipher”. In: *2013 IEEE Symposium on Security and Privacy, SP 2013*. IEEE Computer Society, 2013, pp. 478–492 (cit. on pp. 92–94).

[BBL06] Frederick R. Bieber, Charles H. Brenner, and David Lazer. “Finding Criminals Through DNA of Their Relatives”. In: *Science* 312.5778 (2006), pp. 1315–1316. DOI: `10.1126/science.1122655`. eprint: `http://www.sciencemag.org/content/312/5778/1315.full.pdf` (cit. on p. 93).

[BA10] Marina Blanton and Mehrdad Aliasgari. “Secure Outsourcing of DNA Searching via Finite Automata”. In: *Data and Applications Security and Privacy XXIV, 24th Annual IFIP WG 11.3 Working Conference. Proceedings*. Vol. 6166. Lecture Notes in Computer Science. Springer, 2010, pp. 49–64 (cit. on p. 12).

[BMN17] Erik-Oliver Blass, Travis Mayberry, and Guevara Noubir. “Multi-client Oblivious RAM Secure Against Malicious Servers”. In: *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Proceedings*. Vol. 10355. Lecture Notes in Computer Science. Springer, 2017, pp. 686–707 (cit. on pp. 64, 65).

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. “Divertible Protocols and Atomic Proxy Cryptography”. In: *Advances in Cryptology - EUROCRYPT ’98, International Conference on the Theory and Application of Cryptographic Techniques, Proceeding*. Vol. 1403. Lecture Notes in Computer Science. Springer, 1998, pp. 127–144 (cit. on pp. 19, 50, 51).

[Blo70] Burton H. Bloom. “Space/Time Trade-offs in Hash Coding with Allowable Errors”. In: *Commun. ACM* 13.7 (1970), pp. 422–426 (cit. on p. 16).

[Blu] BlueKrypt. *European Network of Excellence in Cryptology II*. `https://www.keylength.com/en/3/` (cit. on p. 94).

[Bon+13] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. “Private Database Queries Using Somewhat Homomorphic Encryption”. In: *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013. Proceedings*. Vol. 7954. Lecture Notes in Computer Science. Springer, 2013, pp. 102–118 (cit. on p. 38).

[Bos+08] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel H. M. Smid, and Yihui Tang. “On the false-positive rate of Bloom filters”. In: *Inf. Process. Lett.* 108.4 (2008), pp. 210–213 (cit. on pp. 16, 61).

[BCP16] Elette Boyle, Kai-Min Chung, and Rafael Pass. “Oblivious Parallel RAM and Applications”. In: *Theory of Cryptography - 13th International Conference, TCC 2016-A, Proceedings, Part II*. Vol. 9563. Lecture Notes in Computer Science. Springer, 2016, pp. 175–204 (cit. on p. 64).

[BWB09] Marty C. Brandon, Douglas C. Wallace, and Pierre Baldi. “Data structures and compression algorithms for genomic sequence data”. In: *Bioinformatics* (2009) (cit. on p. 99).

[BCP03] Emmanuel Bresson, Dario Catalano, and David Pointcheval. “A Simple Public-Key Cryptosystem with a Double Trapdoor Decryption Mechanism and Its Applications”. In: *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*. Vol. 2894. Lecture Notes in Computer Science. Springer, 2003, pp. 37–54 (cit. on p. 18).

[Bru+08] Fons Bruekers, Stefan Katzenbeisser, Klaus Kursawe, and Pim Tuyls. “Privacy-Preserving Matching of DNA Profiles”. In: *IACR Cryptology ePrint Archive* 2008 (2008), p. 203. URL: `http://eprint.iacr.org/2008/203` (cit. on p. 12).

[Bur00] Samantha Burke. “Delos: Investigating the notion of privacy within the ancient Greek house”. PhD thesis. School of Archaeology and Ancient History, University of Leicester, 2000. URL: `http://hdl.handle.net/2381/8947` (cit. on p. 1).

[CHP92] David Chaum, Eugène van Heijst, and Birgit Pfitzmann. “Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer”. In: *Advances in Cryptology - CRYPTO ’91, 11th Annual International Cryptology Conference, Proceedings*. Vol. 576. Lecture Notes in Computer Science. Springer, 1992, pp. 470–484 (cit. on pp. 19, 58).

[Che+12] Yangyi Chen, Bo Peng, XiaoFeng Wang, and Haixu Tang. “Large-Scale Privacy-Preserving Mapping of Human Genomic Sequences on Hybrid Clouds”. In: *19th Annual Network and Distributed System Security Symposium, NDSS 2012*. The Internet Society, 2012 (cit. on p. 12).

[Cho+] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. “Private Information Retrieval”. In: *J. ACM* 45.6 () (cit. on p. 2).

[CRJ10] Kenneth J. Christensen, Allen Roginsky, and Miguel Jimeno. “A new analysis of the false positive rate of a Bloom filter”. In: *Inf. Process. Lett.* 110.21 (2010), pp. 944–949 (cit. on pp. 16, 61, 94).

[Cur+11] S. Curran et al. “No association between a common single nucleotide polymorphism, rs4141463, in the MACROD2 gene and autism spectrum disorder”. In: *American Journal of Medical Genetics Part B: Neuropsychiatric Genetics* 156.6 (2011), pp. 633–639. ISSN: 1552-485X. DOI: 10.1002/ajmg.b.31201 (cit. on p. 2).

[Dab+10] J. Daborg, M. von Otter, A. Sjölander, S. Nilsson, L. Minthon, DR. Gustafson, I. Skoog, K. Blennow, and H. Zetterberg. “Association of the RAGE G82S polymorphism with Alzheimer’s disease”. In: *Journal of Neural Transmission (Vienna)* 117.7 (2010), pp. 861–867 (cit. on p. 4).

[Dem+17] Daniel Demmler, Kay Hamacher, Thomas Schneider, and Sebastian Stammler. “Privacy-Preserving Whole-Genome Variant Queries”. In: *CANS’17 - 16. International Conference on Cryptology And Network Security*. Lecture Notes in Computer Science. Springer, 2017 (cit. on p. 12).

[Euc56] Euclid. *The thirteen books of Euclid’s Elements. Books III - IX*. Trans. by Thomas Little Heath. Vol. II. Dover, 1956. ISBN: 978-0-486-60089-5 (cit. on p. 13).

[Fra+12] Martin Franz, Peter Williams, Bogdan Carbunar, Stefan Katzenbeisser, Andreas Peter, Radu Sion, and Miroslava Sotáková. “Oblivious Outsourced Storage with Delegation”. In: *Financial Cryptography and Data Security - 15th International Conference, FC 2011. Revised Selected Papers*. Vol. 7035. Lecture Notes in Computer Science. Springer, 2012, pp. 127–140 (cit. on pp. 10, 38, 64, 65).

[Fri09] Keith B. Frikken. “Practical Private DNA String Searching and Matching through Efficient Oblivious Automata Evaluation”. In: *Data and Applications Security XXIII, 23rd Annual IFIP WG 11.3 Working Conference. Proceedings*. Vol. 5645. Lecture Notes in Computer Science. Springer, 2009, pp. 81–94 (cit. on p. 12).

[Fur+17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. “High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority”. In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*. Vol. 10211. Lecture Notes in Computer Science. 2017, pp. 225–255 (cit. on p. 2).

[Gag17] Tommaso Gagliardoni. “Quantum Security of Cryptographic Primitives”. PhD thesis. Darmstadt University of Technology, Germany, 2017 (cit. on pp. 23, 33, 34).

[GKK17] Tommaso Gagliardoni, Nikolaos P. Karvelas, and Stefan Katzenbeisser. “ORAMs in a Quantum World”. In: *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Proceedings*. Vol. 10346. Lecture Notes in Computer Science. Springer, 2017 (cit. on pp. 23, 106).

[Gam85] Taher El Gamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: *Advances in Cryptology, Proceedings of CRYPTO ’84, Proceedings*. Vol. 196. Lecture Notes in Computer Science. Springer, 1985, pp. 10–18 (cit. on pp. 17, 18, 56, 57, 74).

[GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. “TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption”. In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Proceedings*. Vol. 9816. Lecture Notes in Computer Science. Springer, 2016, pp. 563–592 (cit. on pp. 5, 29).

[GHS10] Rosario Gennaro, Carmit Hazay, and Jeffrey S. Sorensen. “Text Search Protocols with Simulation Based Security”. In: *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography*. Vol. 6056. Lecture Notes in Computer Science. Springer, 2010, pp. 332–350 (cit. on p. 12).

[Gol87] Oded Goldreich. “Towards a Theory of Software Protection and Simulation by Oblivious RAMs”. In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, STOC 1987*. ACM, 1987, pp. 182–194 (cit. on pp. 2, 3, 7, 24).

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004 (cit. on pp. 27, 28).

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987*. ACM, 1987 (cit. on p. 12).

[GO96] Oded Goldreich and Rafail Ostrovsky. “Software Protection and Simulation on Oblivious RAMs”. In: *J. ACM* 43.3 (1996), pp. 431–473 (cit. on pp. 7, 8, 24, 25, 43).

[Gol+04] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul F. Syverson. “Universal Reencryption for Mixnets”. In: *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, Proceedings*. Vol. 2964. Lecture Notes in Computer Science. Springer, 2004 (cit. on pp. 74, 87).

[Goo11a] Michael T. Goodrich. “Randomized Shellsort: A Simple Data-Oblivious Sorting Algorithm”. In: *J. ACM* 58.6 (2011), 27:1–27:26 (cit. on p. 43).

[Goo11b] Michael T. Goodrich. “Spin-the-bottle Sort and Annealing Sort: Oblivious Sorting via Round-robin Random Comparisons”. In: *Proceedings of the Eighth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2011*. SIAM, 2011, pp. 93–106 (cit. on p. 43).

[Goo14] Michael T. Goodrich. “Zig-zag sort: a simple deterministic data-oblivious sorting algorithm running in O(n log n) time”. In: *Symposium on Theory of Computing, STOC 2014*. ACM, 2014, pp. 684–693 (cit. on p. 43).

[Goo+12a] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. “Practical oblivious storage”. In: *Second ACM Conference on Data and Application Security and Privacy, CODASPY 2012*. ACM, 2012, pp. 13–24 (cit. on pp. 9, 43).

[Goo+12b] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. “Privacy-preserving group data access via stateless oblivious RAM simulation”. In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*. SIAM, 2012, pp. 157–167 (cit. on pp. 10, 43, 64, 65).

[HT10] Carmit Hazay and Tomas Toft. “Computationally Secure Pattern Matching in the Presence of Malicious Adversaries”. In: *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security. Proceedings*. 2010, pp. 195–212 (cit. on p. 12).

[Hol+12] Andreas Holzer, Martin Franz, Stefan Katzenbeisser, and Helmut Veith. “Secure two-party computations in ANSI C”. In: *the ACM Conference on Computer and Communications Security, CCS’12*. ACM, 2012, pp. 772–783 (cit. on pp. 21, 92, 93).

[Hua+11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. “Faster Secure Two-Party Computation Using Garbled Circuits”. In: *20th USENIX Security Symposium, Proceedings*. USENIX Association, 2011 (cit. on p. 21).

[Hum+15] Mathias Humbert, Kévin Huguenin, Joachim Hugonot, Erman Ayday, and Jean-Pierre Hubaux. “De-anonymizing Genomic Databases Using Phenotypic Traits”. In: *PoPETs* 2015.2 (2015), pp. 99–114 (cit. on p. 12).

[INT] INTEL®. *AES-NI and INTEL® secure key instructions*. `https://software.intel.com/en-us/node/256280` (cit. on p. 2).

[Ish+03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. “Extending Oblivious Transfers Efficiently”. In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Proceedings*. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 145–161 (cit. on p. 21).

[ID03] Anca-Andreea Ivan and Yevgeniy Dodis. “Proxy Cryptography Revisited”. In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2003*. 2003 (cit. on p. 19).

[JSA11] Hallmayer J, Cleveland S, and Torres A. “Genetic heritability and shared environmental factors among twin pairs with autism”. In: *Archives of General Psychiatry* 68.11 (2011), pp. 1095–1102. DOI: `10.1001/archgenpsychiatry.2011.76`. eprint: `/data/Journals/PSYCH/22580/yoa15046_1095_1102.pdf`. URL: `+%20http://dx.doi.org/10.1001/archgenpsychiatry.2011.76` (cit. on p. 2).

[JSS14] Jonathan L. Dautrich Jr., Emil Stefanov, and Elaine Shi. “Burst ORAM: Minimizing ORAM Response Times for Bursty Access Patterns”. In: *Proceedings of the 23rd USENIX Security Symposium*. USENIX Association, 2014, pp. 749–764 (cit. on p. 9).

[KPK17] Nikolaos P. Karvelas, Andreas Peter, and Stefan Katzenbeisser. “Using Oblivious RAM in Genomic Studies”. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2017 International Workshops, DPM 2017 and CBT 2017, Proceedings*. Vol. 10436. Lecture Notes in Computer Science. Springer, 2017 (cit. on pp. 63, 89).

[Kar+14] Nikolaos P. Karvelas, Andreas Peter, Stefan Katzenbeisser, Erik Tews, and Kay Hamacher. “Privacy-Preserving Whole Genome Sequence Processing through Proxy-Aided ORAM”. In: *Proceedings of the 13th Workshop on Privacy in the Electronic Society, WPES 2014*. ACM, 2014, pp. 1–10 (cit. on pp. 37, 89).

[KM10] Jonathan Katz and Lior Malka. “Secure text processing with applications to private DNA matching”. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010*. ACM, 2010, pp. 485–492 (cit. on p. 12).

[Kis+17] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. “Private Set Intersection for Unequal Set Sizes with Mobile Applications”. In: *PoPETs* 2017.4 (2017), pp. 177–197. DOI: 10.1515/popets-2017-0044. URL: https://doi.org/10.1515/popets-2017-0044 (cit. on p. 12).

[KLO12] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. “On the (in)security of hash-based oblivious RAM and a new balancing scheme”. In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*. SIAM, 2012, pp. 143–156 (cit. on pp. 24, 49).

[Lau+13] G. Lauss, C. Schröder, P. Dabrock, J. Eder, K. Hamacher, K.A. Kuhn, and H. Gottweis. “Towards Biobank Privacy Regimes in Responsible Innovation Societies”. In: *Biopreservation and Biobanking* 11 (2013), pp. 319–323 (cit. on p. 2).

[LP09] Yehuda Lindell and Benny Pinkas. “A Proof of Security of Yao’s Protocol for Two-Party Computation”. In: *J. Cryptology* 22.2 (2009), pp. 161–188 (cit. on p. 21).

[Lor+12] Jacob R. Lorch, James W. Mickens, Bryan Parno, Mariana Raykova, and Joshua Schiffman. “Toward Practical Private Access to Data Centers via Parallel ORAM”. In: *IACR Cryptology ePrint Archive* 2012 (2012), p. 133 (cit. on p. 64).

[Maa+13] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. “PHANTOM: practical oblivious computation in a secure processor”. In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*. ACM, 2013, pp. 311–324 (cit. on pp. 3, 9).

[Maf+15] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. “Privacy and Access Control for Outsourced Personal Records”. In: *2015 IEEE Symposium on Security and Privacy, SP 2015*. IEEE Computer Society, 2015, pp. 341–358 (cit. on pp. 3, 11, 64, 65).

[MBM15] Tarik Moataz, Erik-Oliver Blass, and Travis Mayberry. “Constant Communication ORAM without Encryption”. In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 1116 (cit. on p. 24).

[MMB15] Tarik Moataz, Travis Mayberry, and Erik-Oliver Blass. “Constant Communication ORAM with Small Blocksize”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 862–873 (cit. on p. 24).

[Nay+16] Kartik Nayak, Ling Ren, Ittai Abraham, and Benny Pinkas. “An Oblivious RAM with Sublogarithmic Bandwidth Blowup”. In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 849 (cit. on p. 24).

[PM+99] Anne Philippe, Martinez, et al. “Genome-Wide Scan for Autism Susceptibility Genes”. In: *Human Molecular Genetics* 8.5 (1999), pp. 805–812. DOI: 10.1093/hmg/8.5.805. eprint: http://hmg.oxfordjournals.org/content/8/5/805.full.pdf+html (cit. on p. 2).

[PR10] Benny Pinkas and Tzachy Reinman. “Oblivious RAM Revisited”. In: *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference. Proceedings*. Vol. 6223. Lecture Notes in Computer Science. Springer, 2010, pp. 502–519 (cit. on p. 8).

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. “Faster Private Set Intersection Based on OT Extension”. In: *Proceedings of the 23rd USENIX Security Symposium 2014*. 2014, pp. 797–812 (cit. on p. 12).

[Roe13] Lutz Roewer. “DNA fingerprinting in forensics: past, present, future”. In: *Investigative Genetics* 4.1 (2013), p. 22. ISSN: 2041-2223. DOI: 10.1186/2041-2223-4-22 (cit. on p. 92).

[Seb+07] Jonathan Sebat, B. Lakshmi, Dheeraj Malhotra, Jennifer Troge, Christa Lese-Martin, Tom Walsh, Boris Yamrom, Seungtai Yoon, Alex Krasnitz, Jude Kendall, Anthony Leotta, Deepa Pai, Ray Zhang, Yoon-Ha Lee, James Hicks, Sarah J. Spence, Annette T. Lee, Kaija Puura, Terho Lehtimäki, David Ledbetter, Peter K. Gregersen, Joel Bregman, James S. Sutcliffe, Vaidehi Jobanputra, Wendy Chung, Dorothy Warburton, Mary-Claire King, David Skuse, Daniel H. Geschwind, T. Conrad Gilliam, Kenny Ye, and Michael Wigler. “Strong Association of De Novo Copy Number Mutations with Autism”. In: *Science* 316.5823 (2007), pp. 445–449 (cit. on pp. 2, 4).

[Shi+11] Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. “Oblivious RAM with O((logN)3) Worst-Case Cost”. In: *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security,Proceedings*. Vol. 7073. Lecture Notes in Computer Science. Springer, 2011, pp. 197–214 (cit. on pp. 8, 20, 74, 77).

[Sil16] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Springer, 2016 (cit. on p. 17).

[Ste+13a] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. “Path ORAM: an extremely simple oblivious RAM protocol”. In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*. ACM, 2013, pp. 299–310 (cit. on pp. 3, 5, 9, 11, 19, 25, 33, 34, 64, 73, 77, 80).

[Ste+13b] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. “Path ORAM: An Extremely Simple Oblivious RAM Protocol”. In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 280 (cit. on pp. 33, 80, 100).

[SS13a] Emil Stefanov and Elaine Shi. “Multi-cloud oblivious storage”. In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013*. ACM, 2013, pp. 247–258 (cit. on p. 9).

[SS13b] Emil Stefanov and Elaine Shi. “ObliviStore: High Performance Oblivious Distributed Cloud Data Store”. In: *20th Annual Network and Distributed System Security Symposium, NDSS 2013*. The Internet Society, 2013 (cit. on pp. 9, 77).

[SSS12] Emil Stefanov, Elaine Shi, and Dawn Xiaodong Song. “Towards Practical Oblivious RAM”. In: *19th Annual Network and Distributed System Security Symposium, NDSS 2012*. The Internet Society, 2012 (cit. on pp. 9, 77).

[TKC07] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Utku Celik. “Privacy preserving error resilient dna searching through oblivious automata”. In: *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007*. ACM, 2007, pp. 519–528 (cit. on p. 12).

[WCS15] Xiao Wang, T.-H. Hubert Chan, and Elaine Shi. “Circuit ORAM: On Tightness of the Goldreich-Ostrovsky Lower Bound”. In: *Proceedings of the 22nd ACM CCS Conference on Computer and Communications Security, 2015. In Proceedings*. 2015, pp. 850–861 (cit. on p. 10).

[WWC16] Yunling Wang, Jianfeng Wang, and Xiaofeng Chen. “Secure searchable encryption: a survey”. In: *Journal of Communications and Information Networks* 1.4 (2016) (cit. on p. 2).

[WB90] Samuel Warren and Louis Brandeis. “The Right to Privacy”. In: *Harvard Law Review* 4.5 (1890), pp. 193–220 (cit. on p. 1).

[WR13] Lei Wei and Michael K. Reiter. “Ensuring File Authenticity in Private DFA Evaluation on Encrypted Files in the Cloud”. In: *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security. Proceedings*. 2013 (cit. on p. 12).

[WSC08] Peter Williams, Radu Sion, and Bogdan Carbunar. “Building castles out of mud: practical access pattern privacy and correctness on untrusted storage”. In: *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008*. ACM, 2008, pp. 139–148 (cit. on pp. 8, 10, 43).

[WST12] Peter Williams, Radu Sion, and Alin Tomescu. “PrivateFS: a parallel oblivious file system”. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS 2012*. ACM, 2012, pp. 977–988 (cit. on pp. 3, 8, 10, 43, 61, 64).

[Yao86] Andrew Chi-Chih Yao. “How to Generate and Exchange Secrets (Extended Abstract)”. In: *27th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1986, pp. 162–167 (cit. on p. 21).

[ZZQ16] Jinsheng Zhang, Wensheng Zhang, and Daji Qiao. *MU-ORAM: Dealing with Stealthy Privacy Attacks in Multi-User Data Outsourcing Services*. Cryptology ePrint Archive, Report 2016/073. 2016 (cit. on pp. 64, 65).